XRT/table™ for Motif

Programmer's Guide & Reference Manual

UNIX Edition ■ Version 3.0



Software Development Productivity

260 King Street East
Toronto, Ontario, Canada M5A 1K3
(416) 594-1026
www.klgroup.com

January 1999

Part No: KLTM30

Copyright © 1995-1999 by KL Group Inc. All Rights Reserved.

KL Group, the KL Group logo, XRT, XRT/3d, XRT/care, XRT/field, XRT/gear, XRT/graph, XRT/table and XRT PDS are trademarks or registered trademarks of KL Group Inc. in Canada, United States and other countries.

Printed in Canada on recycled paper. 

A Note on XRT Product Licensing

The terms and conditions of the XRT product license are detailed in the license agreement on the next page. Please take a few minutes to read and understand the agreement. The notes below are offered to aid in reading the agreement only, and are not considered part of the agreement itself.

- XRT products (such as XRT/table, XRT/graph) are licensed on a per toolkit (such as Motif) and per architecture (such as SPARC, RS6000, HP 700/800, OpenVMS AXP, etc.) basis.
- There are two types of XRT licenses: single and network-site.
- A single license authorizes use of an XRT product for development of end-user applications on a particular machine identified by a CPU ID. Development activities can include the following:
  - compiling and linking the XRT product library into an end-user application binary.
  - running the XRT Builder tool (if one is supplied).
  - integrating the XRT product into a GUI Builder or UIMS tool. Once an XRT product is integrated into such a tool, you may use the tool for XRT development only on machines that are licensed to use the XRT product.
- A network-site development license authorizes use of an XRT product for development of end-user applications on all similar-architecture machines on the same network and located in the same building.
- You may distribute end-user application binaries including the statically-linked XRT product library anywhere, and as many times as you like, without incurring royalty or run-time fees.

XRT FOR MOTIF BINARY LICENSE AGREEMENT

Important — Read Carefully Before Opening Software Package(s), Installing or Using Software On Your Computer

By opening a sealed software package, or installing, or using this XRT product, you indicate your acceptance of the following XRT for Motif Binary License Agreement. This is a legal agreement between you (either an individual or an entity) and KL Group Inc. If you do not accept and agree with all of the terms of this agreement, promptly destroy any downloaded or installed files, and return the unopened software package(s) and all other materials with proof of payment to your place of purchase, and your license fee will be refunded.

SOFTWARE LICENSE

Please Note: This is a license agreement and **not** an agreement for sale.

Grant of License. This License Agreement ("License") permits you to use one copy of the software you purchased (the "SOFTWARE"), on a single computer after registration of that computer's CPU identification with us. If you have purchased a network site license you may also use the SOFTWARE on computers in the same building having the same architecture as a registered computer, and connected by the same local area network, provided that the total number of such computers, on which the SOFTWARE is ever used, does not exceed the maximum number permitted by the network site license which you ordered. The SOFTWARE is "in use" on a computer when it is loaded into temporary memory (i.e. RAM); when it is installed into permanent memory (e.g. hard disk, CD-ROM, or other storage device); when it is being linked into an end-user application; or when it is accessed from a software development tool such as a compiler, a linker, or a GUI builder. The SOFTWARE is not considered "in use" when installed on a network file server for the sole purpose of file distribution to other computers.

Updates. This agreement will govern the terms of any SOFTWARE updates provided to you by KL Group.

Copyright. The SOFTWARE (including any images, photographs and text incorporated into the SOFTWARE) is owned by KL Group or its suppliers and is protected by Canadian and United States copyright laws and international treaty provisions. Therefore you must treat the SOFTWARE like any other copyrighted material and not reproduce it except that you may make a backup for archival purposes. You may not copy the printed materials accompanying the SOFTWARE.

Other Restrictions. You may not rent, lease, transfer, reverse engineer, decompile, disassemble, create passwords for or translate the Software, except to the extent such foregoing restriction is expressly prohibited by applicable law.

Distribution of applications you build with the SOFTWARE. You have a royalty-free right to reproduce and distribute files in the SOFTWARE that are found in the "lib" directory and which are in a form suitable for linking into object code application files (the "XRT Library Files") provided that (a) you distribute the XRT Library Files only when linked into your software product in such a way as to prohibit further relinking; (b) your software product is not an application development tool that provides similar functionality to the SOFTWARE licensed hereunder; (c) you do not use KL Group's name, logo or trademark to market your software product; (d) you include a valid copyright notice on your software product; and (e) you agree to indemnify, hold harmless, and defend KL Group and its suppliers from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of your software product.

U.S. Government Restricted Rights. The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software — Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is KL Group Inc., 260 King Street East, Toronto, Ontario, Canada, M5A 1K3.

LIMITED WARRANTY

Limited Warranty. KL Group warrants that the SOFTWARE will perform substantially in accordance with the accompanying written materials for a period of sixty (60) days from the date of receipt. Some states/jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. This warranty may not be assigned. KL Group does not exclude product liability or liability for damages resulting out of intent or gross negligence on the part of KL Group or its leading personnel.

Customer Remedies. KL Group's and its suppliers' entire liability and your exclusive remedy shall be, at KL Group's option, either (a) return of the price paid, or (b) repair or replacement of SOFTWARE that does not meet KL Group's Limited Warranty and which is returned to KL Group with a copy of your receipt. This Limited Warranty is void if failure of the SOFTWARE has resulted from accident, abuse, or misapplication. Any replacement SOFTWARE will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

Customer Remedies for Infringement. Should the SOFTWARE become or in KL Group's opinion be likely to become the subject of a claim for infringement of patent, trade secret or copyright, KL Group may (a) procure for you the right to continue to use the SOFTWARE; or (b) replace or modify the SOFTWARE to make it non-infringing, provided that the same function is performed by the replacement or modified SOFTWARE; or if (a) or (b) are not commercially reasonable, (c) terminate the license to use the SOFTWARE, remove the SOFTWARE, and grant you a credit for the license fee as depreciated on a straight-line five (5) year basis.

No Other Warranties or Obligations. To the maximum extent permitted by applicable law, KL Group and its suppliers shall have no other obligations and disclaim all other warranties, either express or implied, including, but not limited to, implied warranties of merchantability and fitness for a particular purpose, with regard to the SOFTWARE and the accompanying printed materials. This limited warranty gives you specific legal rights. You may have others which vary from state/jurisdiction to state/jurisdiction.

No Liability for Consequential Damages. To the maximum extent permitted by applicable laws, in no event shall KL Group or its suppliers be liable for any damages whatsoever (including without limitation, damages for loss of business profits, business interruption, loss of information, or other pecuniary loss) arising out of the use of or inability to use this KL Group product, even if KL Group has been advised of the possibility of such damages. Because some states/jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

Governing Law. If you are a resident of one of the member states of the European Union, this Agreement is governed by the laws of your country of residence, and each of the parties hereto irrevocably attorns to the jurisdiction of the courts of your country of residence. Otherwise, this Agreement is governed by the laws of the Province of Ontario, Canada. Each of the parties hereto irrevocably attorns to the jurisdiction of the courts of the Province of Ontario.

Public Domain Software. This license agreement and the warranties, remedies, limitations, conditions and restrictions contained herein do not apply to any software clearly marked as being in the public domain.

Should you have questions concerning this Agreement, or if you desire to contact KL Group for any reason, please write:

KL Group Inc., 260 King Street East, Toronto, Ontario, Canada, M5A 1K3
Phone (416) 594-1026 / Fax (416) 594-1919

Table of Contents

Preface 1
Introduction 1
New Features in XRT/table 3.0 2
Changes Introduced since XRT/table 2.0 3
Assumptions 5
Typographical Conventions Used in this Manual 5
Related Documents 5
Technical Support 5
Product Feedback and Announcements 6

Part I: Using XRT/table

1 Getting Started: Developing a Simple XRT/table Program 11
1.1 Environment Variables 12
1.2 Compiling and Authorizing a Program 12
1.3 A Basic Table 13
1.4 Setting the Table Size 16
1.5 Adding Column Labels and Formatting the Table 16
1.6 Formatting Parts of the Table 18
1.7 Adding a Title 19
1.8 Proceeding From Here 20

2 XRT/table Basics 21
2.1 Motif Integration 21
2.2 Table Anatomy 23
2.3 Widget Management 24
2.4 Table Contexts 25
2.5 Context and Series Resources 26
2.6 Pointer Resources 28
2.7 XmString Resources and Motif 2.x Restrictions 29
2.8 Colors 29
2.9 Warnings and Error Messages 30

2.10 Using Resource Files 31

3 Programming XRT/table 33

3.1 Resource Summary 33
3.2 Number of Rows/Columns 40
3.3 Visible Rows/Columns 41
3.4 Frozen Rows and Columns 41
3.5 Row and Column Labels 42
Setting and Retrieving Label Values 43
Label Placement and Spacing 44
Label Creation Callbacks 45
3.6 Table Titles 46
3.7 Row Height and Column Width 48
Character Height and Width 49
Pixel Height and Width 50
Variable Height and Width 50
Flexible Rows and Columns 51
Hiding Rows and Columns 51
3.8 Cell and Label Margins 52
3.9 Foreground, Background and Shadow Colors 53
3.10 Cell and Label Text Alignment 55
3.11 Cell and Label Font 55
3.12 Border Types and Sides 56
3.13 Background Pixmap 58
3.14 Text Clipping and Overflow 60
3.15 Cell and Label Spanning 62
3.16 Cell and Label Flashing 63
3.17 User Data 65
3.18 Table Printing 65
Generating PostScript Output 65
Customizing PostScript Fonts 68
Print Callbacks 69

4 XRT/table Data. 73

4.1 The XrtTblCellValues Structure 73
4.2 Getting Data into a Table 76
4.3 Manipulating Table Data 77
Setting and Retrieving Cell Values 77
Adding Rows and Columns 79
Deleting Rows and Columns 80
Moving Rows and Columns 80

Clearing Cell Values 81
Clearing the Entire Table 81
4.4 Using Multiple XrtTblCellValues Structures 81
4.5 Cell Value Callbacks 82
4.6 Getting Data out of the Table 83

5 Programming User Interaction 85

5.1 Default User Interaction 85
5.2 User Interaction and Translation Customization 87
5.3 List Display Mode 88
5.4 Table Scrolling 89
Scrollbars 89
Programming Scrolling 91
Scroll Callbacks 92
5.5 Row and Column Resizing 92
5.6 Cell Traversal 94
5.7 Cell Editing 96
5.8 Cell Selection 99
5.9 Mouse Pointers 104
5.10 Drag and Drop 106

6 Advanced XRT/table Programming 109

6.1 Widgets in Cells/Labels 109
Widgets in a Table 110
Widgets in Resource Files 115
6.2 Cell/Label Value Validation 116
6.3 Using XRT/field with XRT/table 119
6.4 Table Sizing and Resizing 122
6.5 Compound Strings 124
6.6 Series Resource Programming 126
6.7 Multiple Tables with Common Cell Values 128
6.8 XRT/graph and XRT/3d Data 130
6.9 Programming Drag and Drop 130
Drag and Drop Targets 130
Specifying the Text Delimiter 132
Export and Import Callbacks 133
Defining Your Own Targets 134
Data Transfer Format 136
6.10 XRT/table Internationalization 137
6.11 Performance Improvement 137
General Performance 137

Interactive Performance 138
Display Performance 138
Cell Values Handling 139
Managing Contexts Programmatically 139
6.12 Warning and Error Message Customization 142
6.13 Using C++ with XRT/table 142
6.14 Using UIL with XRT/table 142
6.15 Reading and Writing Resource Files 143

Part II: Reference Appendices

A Resource Reference 147
A.1 Widget Synopsis 147
A.2 XRT/table Resources 148
A.3 Constraint Resources 189
A.4 Inherited XRT/table Resources 190

B Procedures and Methods Reference 191

C Macros 221

D Translations and Actions Reference 223
D.1 Default Translations 223
D.2 Action Routines 225

E Data Types 229

F Resource Types and Classes 249

G Resource File and GUI Builder Usage 257
G.1 Syntax 257
G.2 Example Resource File 262

H UIL Arguments 265

I The XrtLabel Widget 271
I.1 Widget Synopsis 271
I.2 Introduction 271
I.3 Using XrtLabel 272
I.4 Use with XRT/table 275
I.5 Resource Reference 276

I.6 Resource Types and Classes 278

J Sample Code 279

J.1 Examples 279
J.2 Demos 280
J.3 simple.c Listing 281

Index 283

Preface

Introduction ■ *New Features in XRT/table 3.0*
Changes Introduced since XRT/table 2.0 ■ *Assumptions*
Typographical Conventions Used in this Manual ■ *Related Documents*
Technical Support ■ *Product Feedback and Announcements*

Introduction

XRT/table Widget

Motif is composed of many different widgets such as buttons, sliders, and menus. Conceptually, XRT/table adds a new widget to Motif. The widget displays rows and columns of text information, pixmaps, and widgets in a scrollable window and can interact with a user.

The table widget has resources that determine how the table will look and behave. Writing programs using XRT/table is very similar to writing any other kind of Motif program; you now have one more Motif widget to work with.

XRT/table may be used in conjunction with KL Group's XRT/graph, XRT/3d and XRT/field widgets. XRT/graph displays 2D data in X-Y plots, bar and pie charts. XRT/3d displays 3D data in surface, contour, and bar chart representations. XRT/field stores text and allows you to convert, complete and validate the text once it is stored.

XRT/table has resources which allow control of:

- Number of rows and columns, both in the table, and visible on the screen.
- Cell contents: cells and labels can contain text values, pixmaps, or widgets.
- Row height and column width.
- Cell/label attributes including border, text alignment, font, and color.
- User interaction including selection methods.
- Cell editability and traversability.
- Compound Strings, which allow cell/label text to use multiple fonts.

XRT/table also provides several procedures and methods which:

- Add, delete, and move table rows and columns.
- Output a representation of the table in PostScript format.

- Read table cell values from an ASCII file, as well as from XRT/graph and XRT/3d datasets.
- Perform data validation of cell values.
- Assist the developer with setting and getting cell ranges.
- Write table cell values to an ASCII file, and in SYLK format.

New Features in XRT/table 3.0

The following list summarizes the major new features introduced in XRT/table 3.0, and indicates where they are discussed in this manual:

Appearance Enhancements

- Set and restore table cursors–table-wide cursors can now be set and returned to their defaults (see *"XrtTblSetAllCursors()"* and *"XrtTblRestoreAllCursors()"* in Appendix B).
- Alignment of Widgets in Cells–a new constraint resource has been added, improving widget alignment in XRT/table cells. Widgets now either adhere to previous resizing rules or adhere to cell's alignment without resizing (see Appendix A).

Programming Enhancements

- Context-based sensitivity–cell sensitivity can now be based on the current context (see Appendix A).
- Improved list display mode–list display mode now more closely resembles the funtionality and behavior of XmList (see section 5.3 on page 88).

Editing Enhancements

- Combo and Spin Boxes –cells can now be edited using combo and spin boxes to enter information from a previously determined list of values known as a picklist (see section 5.7 on page 96).
- Find functions–a range or list of ranges can now be searched by row/column values or by another range value. (see *"XrtTblRangeListFindRange()"* and *"XrtTblRangeListFind"* in Appendix Bfor more information).
- Fast Preview Scrolling–a new resource, when active, displays a small window beside a scrollbar, showing where you would be if you released it.(see Appendix A).

Miscellaneous Improvements

- Improved demos and examples.
- Bug fixes and minor improvements (see the *Release and Installation Notes* document for a listing of the main bugs fixed).

Changes Introduced since XRT/table 2.0

XRT/table 3.0 is backwards-compatible with XRT/table 2.x. If you have existing applications that you want to use with XRT/table 3.0, you should be aware of the following important changes introduced since XRT/table 2.0:

Important Changes	
Recompiling Required	Existing applications must be recompiled (not just relinked) to use this release.
Motif 1.2 or higher Required	You must be running Motif 1.2 (or higher) to be able to use XRT/table 3.0. Motif 2.x is supported.
Cell Values Structure Changed (see section 4.1 on page 73)	A new member, *set_mode*, has been added to the `XrtTblCellValues` structure. This specifies whether cell values are to be stored as Strings or as XmStrings.
Drag and Drop Changed (see section 5.10 on page 106)	The XRT/table drag and drop API has changed. To use XRT/table 2.0 drag and drop capability, set `XmNxrtImportTargets` to NULL when creating the table. (Using XRT/table 2.0 drag and drop is not recommended.)
Widgets Cloned when Location Specified (see section 6.1.1 on page 110)	Widgets are now cloned when the location is specified by `XmNxrtTblLocation`. This means that callbacks cannot be added to the `XmNxrtTblDisplayWidgetCallback` list after the location is specified.
ValidateCell Callback Changed (see "XmNxrtTblValidateCellCallback" on page 185)	The `XRTTBL_REASON_VALIDATE_CELL` callback reason (in the `XrtTblValidateCellCallbackStruct` callback structure) has been changed to `XRTTBL_REASON_VALIDATE_CELL_BEGIN`.

The following features have also been changed:

Enumerated Types are now unsigned chars	All XRT/table enumerated types are now of type `unsigned char` (to conform to OSF standards). This has no effect unless you have been assuming that they are of type `int`.
Cell/Label Border around Widgets is not default	XRT/table now only draws a border around a cell or label containing a widget if the widget's `XmNborderWidth` has been set to a non-zero value. To preserve the previous appearance of your tables, set `XmNborderWidth` to 1.
XmNxrtTblLabelCreateFunc has been superseded (see section 3.5.3 on page 45)	`XmNxrtTblLabelCreateFunc` has been superseded by `XmNxrtTblLabelCreateCallback`. `XmNxrtTblLabelCreateFunc` is supported in this version of XRT/table.
XmNxrtTblPSFontListContext and XmNxrtTblPSFontListSeries are now obsolete (see section 3.18 on page 65)	`XmNxrtTblPSFontListContext` and `XmNxrtTblPSFontListSeries` are now obsolete, and have been replaced by `XmNxrtTblPSFontMapList`. These resources are supported in this version of XRT/table.
XmNxrtTblCellValueFunc has been superseded (see section 4.5 on page 82)	`XmNxrtTblCellValueFunc` has been superseded by `XmNxrtTblCellValueCallback`. `XmNxrtTblCellValueFunc` is supported in this version of XRT/table.
XmNxrtTblRepaint Changed (see "XmNxrtTblRepaint" on page 177)	`XmNxrtTblRepaint` now expects values of type `XrtTblRepaint`. Valid values are `XRTTBL_REPAINT_ON` (repaint), `XRTTBL_REPAINT_OFF` (don't repaint, but perform size calculations), and `XRTTBL_REPAINT_NOCALC` (don't repaint or perform calculations).

XrtTblCreateWidgetCallback-Struct Changed (see "XrtTblCreateWidgetCallbackStruct" on page 234)	An *event* field has been added to conform to standard Motif specifications.
XrtTblDisplayWidgetCallback-Struct Changed (see "XrtTblDisplayWidgetCallbackStruct" on page 234)	An *event* field has been added to conform to standard Motif specifications.
XrtTblResizeCallbackStruct Changed (see "XrtTblResizeCallbackStruct" on page 243)	An *event* field has been added to conform to standard Motif specifications.
XrtTblValidateCellCallback-Struct Changed (see "XrtTblValidateCellCallbackStruct" on page 246)	An *event* field has been added to conform to standard Motif specifications.
XrtTblScrollCallbackStruct Changed (see "XrtTblScrollCallbackStruct" on page 229)	Two new members have been added to this structure: the new value of the `XmNxrtTblTopRow` or `XmNxrtTblLeftColumn` resource, which is an int value, and the text that would be displayed in the preview window, which is a string.

Assumptions

This manual assumes that you are proficient with the C language and the Motif toolkit. C concepts such as "an array of `char *`", and "pointer to a structure" must be understood before beginning to program Motif and XRT/table applications. An understanding of basic Motif concepts such as getting and setting resource values is required before continuing with this manual. See page 5 for information on Motif documentation.

Typographical Conventions Used in this Manual

`Typewriter Font`	■ C language source code and examples of file contents. ■ XRT/table and Motif resource names and constants. ■ Commands that you enter at a terminal window.
Italic Text	■ Pathnames, filenames, program, procedure and parameter names. ■ New terms as they are introduced, and to emphasize important words. ■ Figure and table titles. ■ The names of other documents referenced in this manual, such as the *Motif Programming Manual*.

Related Documents

- *Motif Programming Manual* and *Motif Reference Manual* from O'Reilly & Associates, Inc.
- *X Toolkit Intrinsics Reference Manual*, *Xlib Programming Manual* and *Xlib Reference Manual* from O'Reilly & Associates, Inc. These manuals are not required to develop applications using XRT/table and Motif; however, an understanding of Xlib can be useful for X Window application developers.

Technical Support

KL Group provides an update and technical support package called XRT/care. Sixty days of XRT/care coverage are included from the date you purchased XRT/table. You may purchase annual XRT/care agreements from your XRT Reseller or KL Group. XRT/care ensures you will receive all XRT/table bug-fix releases and upgrades. It also entitles you to receive help with installation problems, programming problems or other technical issues.

If you purchased XRT/table from KL Group, contact KL Group for technical support; if you purchased XRT/table from an XRT reseller, please contact the reseller for technical support. The technical support contact information is on your password letter.

Many of the initial problems you may encounter are basic installation or setup problems. Consult the "Troubleshooting" section in the *Release and Installation Notes* accompanying this release for solutions to common installation and setup problems.

Any request for support **must** include your XRT/table license serial number. Collecting the following additional information will help us serve you better:

- Identification – your name, company name, case number (if in regards to a previous case)
- Product – XRT/table version and architecture
- Platform – the version of the operating system, X, and Motif you are using
- Problem – reproducable description of the problem you are having

Online Support Resouces

KL Group's web site provides additional avenues for technical support, for example:

XRT Technical Support: (general info and links)	`http://www.klgroup.com/cs/tech/xrt/`
XRT Knowledge Base:	`http://www.klgroup.com/cgi-bin/xrt_search.cgi`
XRT Product FAQs:	`http://www.klgroup.com/cs/tech/xrt/faq/`

Product Feedback and Announcements

We are interested in hearing about how you use XRT/table, any problems you encounter, or any additional features you would find helpful. The majority of enhancements to XRT products are the result of customer requests.

Please send your comments to:

KL Group Inc.
260 King Street East
Toronto, Ontario, M5A 1K3 Canada

Phone: (416) 594-1026
Fax: (416) 594-1919
Email: `dev_xrt@klgroup.com`

Internet: `news://news.klgroup.com/klg.forum.xrt`

While we appreciate your feedback, we cannot guarantee a response. Please do not use the `dev_xrt` email address for technical support questions.

Occasionally, we send XRT-related product announcments to our customers using an email list. To add yourself to this mailing list, send email with the word "`subscribe`" in the body of the message to `xrt_announce-request@klgroup.com`. Visit the KL Group web site at `http://www.klgroup.com` for more details.

Part I

Using XRT/table

1

Getting Started: Developing a Simple XRT/table Program

Environment Variables ■ *Compiling and Authorizing a Program*
A Basic Table ■ *Setting the Table Size*
Adding Column Labels and Formatting the Table
Formatting Parts of the Table ■ *Adding a Title* ■ *Proceeding From Here*

This chapter allows you to immediately try out XRT/table by compiling and running an example program, introducing some fundamental XRT/table programming concepts. This application displays information about employees in a fictitious company. The finished program, *simple.c*, is located in *$XRTHOME/src/table/examples*, and is listed in Appendix J on page 279.

The data to be displayed appears below:

Name	Address	Phone Number	Position	Salary
James Q. Doohan	1701 Planetia Blvd. Anytown, U.S.A.		President	$245,000
John F. Kricfalusi	983 Nickelodeon Street Anytown, U.S.A.	(999) 555-9876	Animator	$10,000
Marc Lenard	6 Gene Crescent Anytown, U.S.A.	(999) 555-1212	Telemarketer	$10 / hr.
Hikaru I. Takei	134 Adelaide Street E. Suite 204 Anytown, U.S.A.	(999) 594-1026	OEM Sales	50%
Melissa A. Truman	475 Woodview Line Anytown, U.S.A.	(999) 555-9030	Technical Writer	$50,250
Stephanie L. Truman	388 Appleby Road Anytown, U.S.A.	(999) 555-2642	System Analyst	$85,750

Name	Address	Phone Number	Position	Salary
Bill West	1001 Spumco Way Anytown, U.S.A.	(999) 555-9966	Announcer	$17,500

1.1 Environment Variables

Before getting started, be sure your Unix environment is set up correctly. An environment variable called *XRTHOME* should be defined as the name of the directory containing the XRT/table distribution files.

You can determine the current value for *XRTHOME* by entering:

```
echo $XRTHOME
```

If your *XRTHOME* environment variable is not set up correctly, see your system administrator or the *Release and Installation Notes*.

Before attempting to use XRT/table, you must be able to compile Motif applications. Consult your Motif documentation to determine where the Motif library and include files are located. Make sure you can compile and run simple Motif applications before you begin to use XRT/table.

1.2 Compiling and Authorizing a Program

If the X and Motif libraries and include files are installed in the standard places (normally */usr/lib*, */usr/include/X11*, and */usr/include/Xm)*, *simple.c* may be compiled and authorized by entering:[1]

```
cc simple.c -I$(XRTHOME)/include \
    -L$(XRTHOME)/lib -o simple \
    -lxrttable -lXm -lXpm -lXt -lX11
$XRTHOME/bin/xrt_auth simple
```

If the libraries and include files are not installed in standard locations, additional use of the *-I* and *-L* compiler flags will be required.

xrt_auth

After the XRT/table library has been linked into an executable application, it must be authorized with the *xrt_auth* utility. This utility will check that the machine you are building on is authorized to use XRT/table, and will enable the application. *xrt_auth* takes two optional switches, `-v` for verbose messages and `-q` for quiet mode (that is, no messages):

```
xrt_auth [-q | -v] executable_file
```

1. Compilation options depend on your computer architecture. See the *Makefile* in *$XRTHOME/src/table/examples* for an example of how to compile on your system.

Once an end-user application has been enabled by *xrt_auth*, it may be executed on any machine, anywhere. There are no special runtime requirements for applications built with XRT/table.

If your application is not successfully enabled using *xrt_auth*, table widgets will display for a few seconds followed by the message shown in Figure 1. A more detailed message with troubleshooting tips is also written to *stdout*.

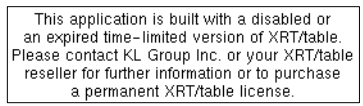


Figure 1 Message Displayed by an Expired or Non-authorized Application

Note: For information on linking and authorizing shared libraries, see the note in the release notes.

C++ and ANSI C

The XRT/table include files are compatible with C++ and ANSI C as well as with traditional "Kernighan and Ritchie" C.

1.3 A Basic Table

To work up to *simple.c*, we start with *table1.c*, listed in Figure 3. This is a minimal C program that loads the text data from a file, creates a Motif shell widget and then creates a table widget that displays the text. When compiled and run, the window shown in Figure 2 displays.

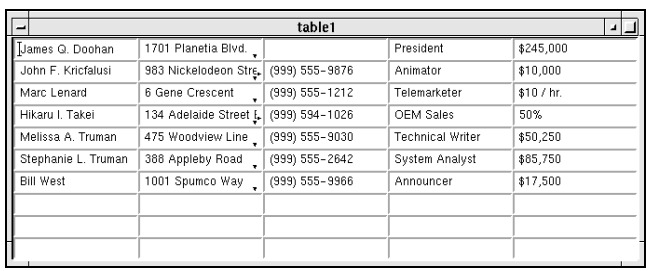


Figure 2 The window displayed by table1.c

Most of the code should be familiar to Motif programmers. The statement on line 9 allocates space for the `XrtTblCellValues` structure (pointed to by `values`). Line 22 loads that structure with the text data from the file *simple.dat*. Line 19 initializes the Intrinsics and creates a toplevel shell in the standard Xt Intrinsics fashion.

Lines 23–27 use the standard Xt Intrinsics procedure *XtVaCreateManagedWidget()* to instantiate a table widget that is a child of `toplevel`. All table resources except

XmNxrtTblCellValues and XmNxrtTblXmSetStringMode are left at their default values.

If you are linking to Motif 2.x, you will need to set the XmNxrtTblXmStringSetMode resource to specify whether you want to set and get XmString values as strings or XmStrings. This resource can only be set when creating the table.

Standard Xt methods (like *XtSetValues()*, *XtGetValues()*, *XtVaSetValues()*) may be used to manipulate the table widget. XRT/table is subclassed from the Motif Manager class, so in addition to having table resources, it also has Manager, Constraint, Composite, and Core resources. For example, it is possible to set the XmNbackground resource of a table.

End-User Interaction

More than simply a display tool, XRT/table is an interactive widget. By default, end-users can interact with the widget in the following ways:

- Editing cells–XRT/table uses a standard XmText widget to enable the user to edit the cell that currently has the input focus (called the *current cell*).
- Resizing rows/columns–A user can interactively change the width of columns and the height of rows.
- Scrolling the table–When a table is larger than its window, XRT/table uses two standard XmScrollBar widgets to enable the user to scroll through the entire table.
- Selecting cells–A user can interactively select cells, although this capability is disabled by default.
- Drag and drop–A user can select text and move or copy it to another portion of the table, or copy in text from other tables or other XtSelections.

```
1   #include <Xm/XrtTable.h>
2
3   int
4   main(argc, argv)
5   int argc;
6   char *argv[];
7   {
8       Widget toplevel, table;
9       XrtTblCellValues *values;
10      FILE *fp;
11
12      /* Open data file */
13      if (!(fp = fopen("simple.dat", "r"))) {
14         perror("simple.dat");
15         exit(0);
16      }
17
18      /* Initialize Intrinsics */
19      toplevel = XtInitialize(argv[0], "Simple", NULL, 0, &argc, argv);
20
21      /* Read data values, and load into table */
22      values = XrtTblReadAscii(toplevel, fp, ' , ');
23      table = XtVaCreateManagedWidget("table",
24                                  xtXrtTableWidgetClass, toplevel,
25                                  XmNxrtTblCellValues,   values,
26                                  NULL);
27      XrtTblDestroyCellValues(values);
28      fclose(fp);
29
30      XtRealizeWidget(toplevel);
31      XtMainLoop();
32  }
```

Figure 3 table1.c listing

Loading Cell Values

A common task in any XRT/table program is to load the text data (called *cell values*) into a format that XRT/table understands (the `XrtTblCellValues` structure). To allocate and load cell values from a file, use the *XrtTblReadAscii()* method.[1]

Figure 4 lists the contents of the *simple.dat* file used in this example. This is an ASCII file containing text structured in the format that *XrtTblReadAscii()* understands. Each line contains the cell values for one row, each column separated by commas. Special characters are handled by adding a backslash character (for example, use "`\,`" to include a comma as part of the text).

1. A more general way is to simply populate XRT/table's default `XrtTblCellValues` structure using C code.

```
James Q. Doohan,1701 Planetia Blvd.\nAnytown\, U.S.A.,,President,$245\,000
John F. Kricfalusi,983 Nickelodeon Street\nAnytown\, U.S.A.,(999) 555-9876,Animator,$10\,000
Marc Lenard,6 Gene Crescent\nAnytown\, U.S.A.,(999) 555-1212,Telemarketer,$10 / hr.
Hikaru I. Takei,134 Adelaide Street E.\nSuite 204\nAnytown\, U.S.A.,(999) 594-1026,OEM Sales,50%
Melissa A. Truman,475 Woodview Line\nAnytown\, U.S.A.,(999) 555-9030,Technical Writer,$50\,250
Stephanie L. Truman,388 Appleby Road\nAnytown\, U.S.A.,(999) 555-2642,System Analyst,$85\,750
Bill West,1001 Spumco Way\nAnytown\, U.S.A.,(999) 555-9966,Announcer,$17\,500
```

Figure 4 The simple.dat file

1.4 Setting the Table Size

The table shown in Figure 2 has three blank rows at the end; if you don't specify the number of rows and columns in the table, it is set to 10 rows and 5 columns by default.

All resources for a table can be specified when you create the table, or they may be changed at any time as the program runs using *XtSetValues()* or *XtVaSetValues()*. You can ask for the current value of any resource with *XtGetValues()* or *XtVaGetValues()*.[1]

Since there are seven employees to display in this example, we will add the following lines to the *XtVaCreateManagedWidget()* call to set the `XmNxrtTblNumRows` and `XmNxrtTblNumColumns` resources:[2]

```
XmNxrtTblNumRows,       7,
XmNxrtTblNumColumns,    5,
```

1.5 Adding Column Labels and Formatting the Table

The table displayed by the *table1.c* program is not very useful to an end-user. There are no column labels, and the table needs some formatting.

Setting Labels

To add column labels to the table, we will add the following line to the *XtVaCreateManagedWidget()* call:

```
XmNxrtTblColumnLabels,      &labels,
```

1. The values of many table resources can change quite frequently, usually as a result of user interaction with the widget.

2. The relationship between the rows/column in the table and the `XrtTblCellValues` structure is slightly more complex than this. See chapter 4, "XRT/table Data", on page 73, for complete details.

This resource takes an `XrtTblXmStringList` structure as its value. We need to load the text for the column labels into a `XrtTblXmStringList` structure, as shown by the following code:

```
XrtTblXmStringList labels;

static String label_list[] =
    { "Name", "Address", "Phone Number",
      "Position", "Salary" };
labels.length = XtNumber(label_list);
labels.list = (XmString *)label_list;
```

Resizing to Display all Text

The table shown in Figure 2 does not display all of the text in the Address column. Small *clipping arrows* appear at the bottom-right corner of cells that contain more text than is currently displayed.

Although you could let the user resize the rows/columns manually to see all of the text, it's often better to ensure that all of the text you set for each cell is fully displayed at the outset. XRT/table can automatically resize rows and columns to fit the text they contain by adding the following to the *XtVaCreateManagedWidget()* call:

```
XmNxrtTblPixelWidthContext,     XRTTBL_VARIABLE,
XmNxrtTblPixelHeightContext,    XRTTBL_VARIABLE,
```

Overall Table Formatting

Figure 5 shows the result of combining the above changes with setting table background/foreground colors. The following call creates the new table:

```
table = XtVaCreateManagedWidget("table",
            xtXrtTableWidgetClass,      toplevel,
            XmNxrtTblNumRows,           7,
            XmNxrtTblNumColumns,        5,
            XmNxrtTblCellValues,        values,
            XmNxrtTblColumnLabels,      &labels,
            XmNxrtTblBackgroundContext, "honeydew1",
            XmNxrtTblForegroundContext, "blue4",
            XmNxrtTblPixelWidthContext, XRTTBL_VARIABLE,
            XmNxrtTblPixelHeightContext,XRTTBL_VARIABLE,
            NULL);
```



Figure 5 table1 with basic improvements

1.6 Formatting Parts of the Table

Although the table is now more useful, it would look better if the column labels looked different from the cells. We will also highlight the President's salary.

When you want to change attributes for *part* of a table, the *XtVaSetValues()* call must also set a special table resource—`XmNxrtTblContext`. This resource specifies the part of the table (called the *context*) that you want to set attributes for.[1] Resources that apply to the current context are called Context resources; their names always end with "Context".

If you set Context resources without setting a context (using *XrtTblSetContext()*), XRT/table uses the last context set. Since *table1.c* did not set a context, the Context resources were applied to the entire table (the default value of `XmNxrtTblContext`).

The following code sets the context to all the column labels, and then sets colors and alignment for that context:

```
XtVaSetValues(table,
    XmNxrtTblContext,XrtTblSetContext(XRTTBL_LABEL, XRTTBL_ALL),
    XmNxrtTblBackgroundContext,  "blue",
    XmNxrtTblForegroundContext,  "white",
    XmNxrtTblAlignmentContext, XRTTBL_ALIGNMENT_TOPCENTER,
    NULL);
```

The following code sets the context to the cell containing the President's salary, and then sets it to a different foreground color:

```
XtVaSetValues(table,
    XmNxrtTblContext,            XrtTblSetContext(0,4),
    XmNxrtTblForegroundContext,  "red",
    NULL);
```

1. See the next chapter for a complete discussion of *contexts*.

1.7 Adding a Title

Most tables are much more useful if they have a title. To put a title on a table, we add an XrtLabel widget to the top of the table widget, above the column labels.[1] The following call does this and sets some XrtLabel resources:

```
title = XtVaCreateWidget("title",
    xtXrtLabelWidgetClass,      table,
    XmNxrtLblBorderType,        XRTLBL_BORDER_SHADOW,
    XmNlabelString,             "Employee Listing",
    XmNxrtTblLocation,          XRTTBL_LOCATION_TOP,
    XmNxrtTblTopOffset,         10,
    XmNxrtTblBottomOffset,      20,
    XtVaTypedArg,               XmNfontList,
    XmRString,
    "-*-times-bold-r-*-*-*-180-*-*-*-*-iso8859-1",
    strlen("-*-times-bold-r-*-*-*-180-*-*-*-*-iso8859-1")+1,
    NULL);
```

All of the improvements to *table1.c* made in this chapter are combined in *simple.c*. This program is located in *$XRTHOME/src/table/examples*, and is listed in Appendix J on page 279. When *simple.c* is compiled and executed, the window shown in Figure 6 displays.

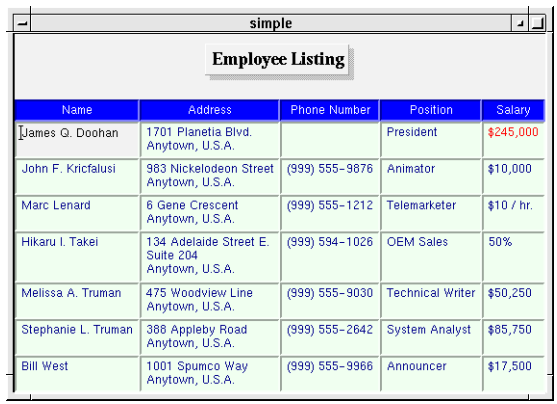


Figure 6 The simple.c window

1. XRT/table can actually place any Motif widget at the top of the table. The XrtLabel widget is a Primitive widget designed specifically to add attractive titles to tables. For more information on XrtLabel, see Appendix I on page 271.

1.8 Proceeding From Here

The following suggestions should help you be productive with XRT/table as quickly as possible:

- Chapter 2 contains basic XRT/table concepts. You should read this chapter to get a good grounding in XRT/table.
- The examples and demos are useful learning aids. You can see what XRT/table can do, and how to do it, by running them and examining their source code. Appendix J on page 279 briefly describes all of the demos and examples.

2

XRT/table Basics

Motif Integration ■ *Table Anatomy*
Widget Management ■ *Table Contexts*
Context and Series Resources ■ *Pointer Resources*
XmString Resources and Motif 2.x Restrictions ■ *Colors*
Warnings and Error Messages ■ *Using Resource Files*

The XRT/table widget is a scrollable window that displays text, pixmaps, or widgets in rows and columns. XRT/table also automatically handles interaction with a user: resizing rows/columns, text editing, scrolling, and cell selection.

2.1 Motif Integration

Conceptually, XRT/table adds a new widget to the Motif toolkit. A Motif programmer manipulates this widget using the familiar Xt Intrinsics calls.

The Xt Intrinsics defines an object-oriented architecture that organizes widgets into classes. For example, the Motif XmPushButton widget is a subclass of XmLabel, which is a subclass of XmPrimitive, which is subclassed from the Xt Core widget. XRT/table is subclassed from XmManager. This means that XmManager and Core resources can be set in addition to the resources defined by the XRT/table widget. For example, you can set the default background color with the `XmNbackground` resource.

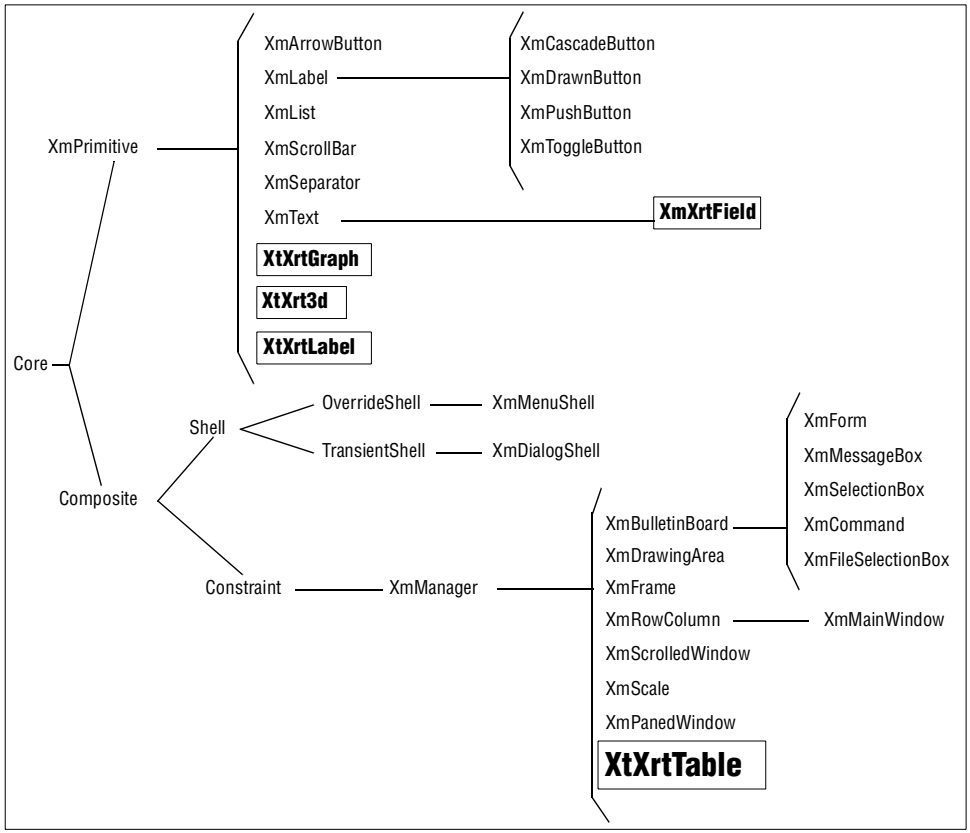


Figure 7 XRT widgets' position within the Motif Class Hierarchy

2.2 Table Anatomy

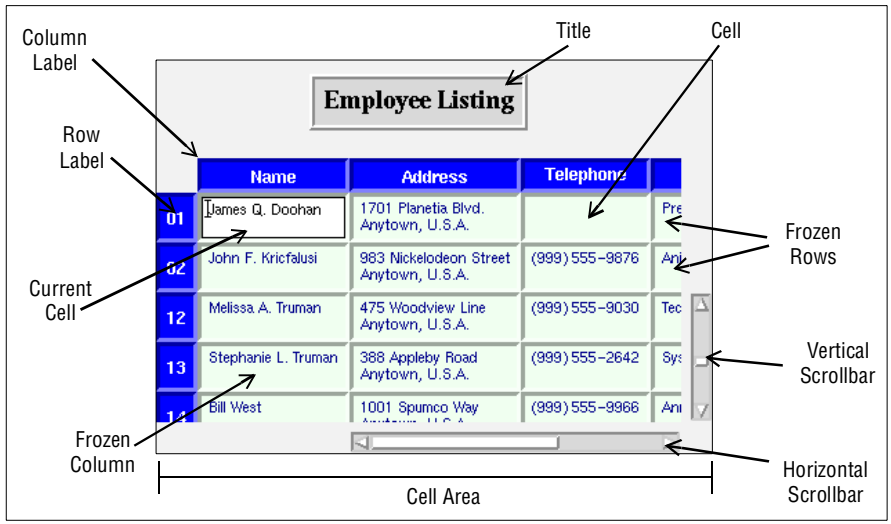


Figure 8 Components of a Table

XRT/table provides a scrollable viewing area into its cells and labels. There are often far more cells than there is room to display, similar in concept to the XmScrolledWindow widget. The following list defines the components that are new to Motif programmers:

Cell
An individual container of table data. A cell can contain text or a widget. It can also contain a background pixmap. A cell is *visible* if it is currently scrolled into view. Cell values are stored in an `XrtTblCellValues` structure. The entire collection of displayed cells is called the *cell area.*

Label
A non-editable cell appearing in a row at the top of the table, or in a column at the left side of a table. Like cells, labels can contain text or a widget, and can display a pixmap in the background. Unlike cells, the label row and column cannot be scrolled out of view. A label is visible if it is currently scrolled into view. Label values are stored in two `XrtTblXmStringList` structures, separate from cell values; one for row labels, one for column labels.

Current Cell
The cell that currently has the user input focus. End-users can enter and edit the value of this cell (unless you disable this ability).

2.3 Widget Management

XRT/table creates and manages several child widgets for its own internal use. Three are standard Motif widgets (shown in Figure 9)–one XmText widget and two XmScrollBar widgets. XRT/table also creates and manages several specialized private widgets that clip the cell/label viewing areas.[1]

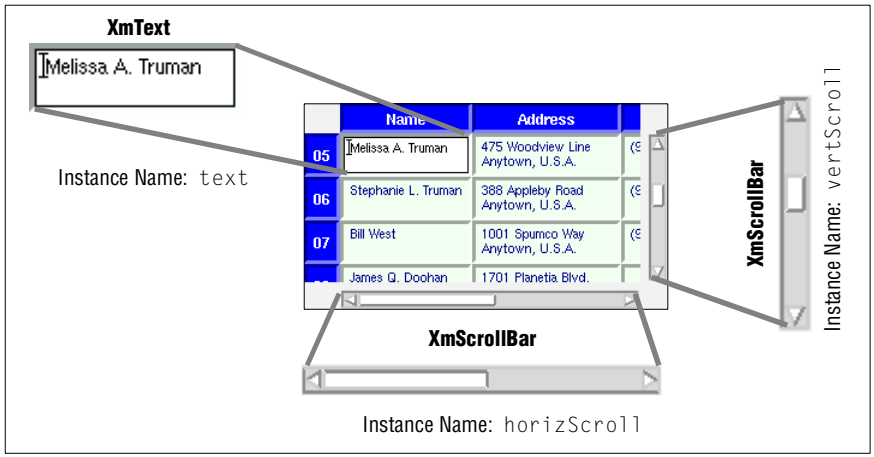


Figure 9 Standard Motif widgets used by XRT/table

XmText widget

This widget is the current cell. It provides end-users with the ability to enter and edit cell values. As a user traverses from cell to cell, XRT/table unmaps the XmText widget and maps it over the next cell.

XmScrollBar widgets

These widgets are created and displayed if the number of rows or columns in the table is greater than the number of rows or columns visible on the screen. They provide end-users with the ability to scroll through the entire table.

You can access and set the resources of these widgets in your application.

1. The XmScrolledWindow widget similarly uses a private clip window widget to constrain the view of the work window (which the user manipulates with the scrollbars).

2.4 Table Contexts

A context is composed of a row and column index, both zero-based. The *current context* specifies the portion of a table's cells and labels that an application sets and retrieves attributes for. `XmNxrtTblContext` defines the current context.

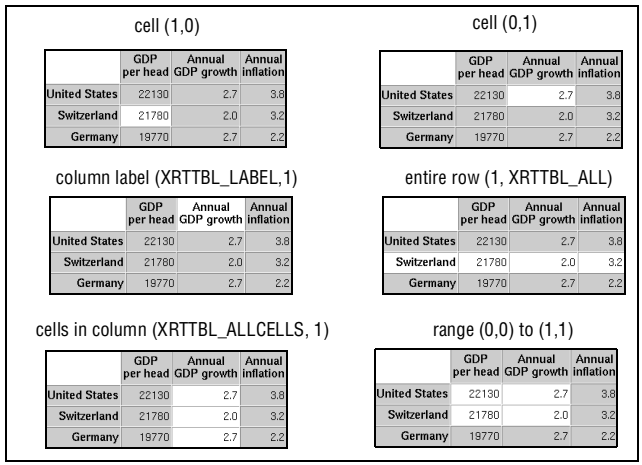


Figure 10 Cell and Label Referencing

The following describes the types of table contexts you can reference:

A Cell

Referenced by a row index and a column index, starting from zero. For example, the location of the second cell in the second row is `(1, 1)`.

A Row Label

Referenced by a row index and the constant `XRTTBL_LABEL`. For example, the second row's label is located at `(1, XRTTBL_LABEL)`. To reference all row labels, use `(XRTTBL_ALL, XRTTBL_LABEL)`.

A Column Label

Referenced by the constant `XRTTBL_LABEL` and a column index. For example, the second column's label is located at `(XRTTBL_LABEL, 1)`. To reference all column labels, use `(XRTTBL_LABEL, XRTTBL_ALL)`.

All Cells in Row/Column

Referenced by the constant `XRTTBL_ALLCELLS`. For example, to reference all of the cells in the second row, use `(1, XRTTBL_ALLCELLS)`.

Entire Row/Column

Referenced by the constant `XRTTBL_ALL` and includes the label. For example, to reference the entire second column, use `(XRTTBL_ALL, 1)`.

All Cells in Table
Referenced by (XRTTBL_ALLCELLS, XRTTBL_ALLCELLS).

Entire Table
Referenced by (XRTTBL_ALL, XRTTBL_ALL) and includes the labels.

Range
Referenced by the location of the top-left cell/label and the location of the bottom-right cell/label in the range. A range is a block of cells and/or labels, and can be referenced as one context.

2.5 Context and Series Resources

Resources that work with the current context are called *Context resources.* All Context resources have a corresponding *Series resource*, for example, XmNxrtTblBackgroundContext and XmNxrtTblBackgroundSeries.

The Series version of the resource specifies the entire series of resource settings over the entire table. Both versions affect the same internal list of attribute settings over the entire table. If you set a Context resource, its corresponding Series resource is also updated. In most cases there is no appreciable performance difference between using Context and Series resources.[1]

Context resources are designed for use by application programs. Their advantage is that once a context has been set, any Context resource set is applied to that context. This makes it easy to set several table attributes for the same portion of the table. This manual uses Context resources in discussions and examples because they are the primary programmatic interface to the table's cells and labels.

Series resources are designed to be used in resource files and fallback resources. Their primary advantage is that the context is included for the setting of each member in the series. One attribute can be set over several contexts in a resource file. However, they are difficult to program and manage because a series contains all settings made over the entire table.

Setting Context Resources
One of two resources specifies the current context. Use XmNxrtTblContext to set most types of contexts. As a convenience, you can set a context to all the cells in a range using XmNxrtTblRangeContext. When you set one of these two resources, the other is ignored.

1. If you are setting the same resource over many individual cells and contexts, it is faster to use the Series resource.

Use the *XrtTblSetContext()* procedure to set most types of contexts (using XmNxrtTblContext):

```
XtVaSetValues(table,
    XmNxrtTblContext,     XrtTblSetContext(XRTTBL_LABEL,
                            XRTTBL_ALL),
    XmNxrtTblBackgroundContext,  "grey",
    XmNxrtTblForegroundContext,  "black",
    NULL);
```

Use the *XrtTblSetRange()* procedure to set a context to a range (using XmNxrtTblRangeContext):

```
XtVaSetValues(table,
    XmNxrtTblRangeContext,       XrtTblSetRange(1,1,5,5),
    XmNxrtTblBackgroundContext,  "grey",
    XmNxrtTblForegroundContext,  "black",
    NULL);
```

Getting Context Resources

To retrieve a Context resource value for a context that is different than the current context, set the context before calling *XtVaGetValues()*:

```
XtVaSetValues(table,
    XmNxrtTblContext,           XrtTblSetContext(2,2),
    NULL);
XtVaGetValues(table,
    XmNxrtTblBackgroundContext, &bg_color,
    NULL);
```

Resetting Context Resources

To reset a Context resource to its default value, use XRTTBL_REMOVE:

```
XtVaSetValues(table,
    XmNxrtTblRangeContext,      XrtTblSetRange(1,1,5,5),
    XmNxrtTblBackgroundContext, "grey",
    NULL);
XtVaSetValues(table,
    XmNxrtTblRangeContext,       XrtTblSetRange(1,1,5,5),
    XmNxrtTblBackgroundContext,  XRTTBL_REMOVE,
    NULL);
```

The second call to *XtVaSetValues()* undoes the effects of the first.

Other context-setting operations are unaffected by XRTTBL_REMOVE:

```
XtVaSetValues(table,
    XmNxrtTblRangeContext,       XrtTblSetContext(2,2),
    XmNxrtTblBackgroundContext,  "green",
    NULL);
XtVaSetValues(table,
    XmNxrtTblRangeContext,       XrtTblSetRange(1,1,5,5),
    XmNxrtTblBackgroundContext,  XRTTBL_REMOVE,
    NULL);
```

In this case, the cell (2,2) still has its background color set to green.

If XRTTBL_REMOVE is specified for a context which has not been set, it is ignored.

Setting Series Resources

A Series resource specification lists the entire series. In a resource file, each member contains a context and a value surrounded by parentheses. The following resource file fragment sets background colors for several contexts:

```
*.xrtTblBackgroundSeries: (ALL ALL gray) (2 2 blue) \
    (2 2 red) (ALL 4-7 yellow) (ALL 4 red) \
    (LABEL ALL yellow) (LABEL 2 red) (LABEL 5 black) \
    (ALL LABEL yellow) (1 LABEL red)
```

Series settings are processed in the order they are read, so you can see that although the `(LABEL ALL yellow)` member sets the background color of all column labels to yellow, subsequent members (`(LABEL 2 red)` and `(LABEL 5 black)`) override this. Consult Appendix G on page 257 for full details on specifying resources in resource files.

Programming Series Resources

Programming Series resources is complex and usually not necessary. See section 6.6 on page 126 for a complete discussion.

2.6 Pointer Resources

Some XRT/table resources return pointers to structures or strings when used in an *XtGetValues()* call. Since this is a pointer to the widget's internal storage, an application should *never* use this pointer to change any data in memory, and the pointer should never be altered or freed. Everything the pointer points to should be considered *read only* and *temporary*.

XRT/table makes its own internal copy of all data passed to it through a pointer resource.

The XRT/table duplicating procedures (such as *XrtTblSeriesCopy()*) and Xt procedures like *XtNewString()* can be used to make copies of some types of data. XRT/table free procedures (such as *XrtTblSeriesDestroy()*) and Xt procedures like *XtFree()* can be used to release these copies. Appendix B on page 191 has a complete list of XRT/table procedures and methods.

The following example illustrates copying a returned pointer:

```
XmString value1;

XtVaSetValues(table,
        XmNxrtTblContext, XrtTblSetContext(0,0),
        NULL);
XtVaGetValues(table,
        XmNxrtTblCellValueContext, &value1,
        NULL);
value1 = XmStringCopy(value1);
```

2.7 XmString Resources and Motif 2.x Restrictions

If you are using Motif 1.2, a String can be used wherever an XmString is expected. (Motif 2.x does not allow this.) XmStrings are used in the following places:

- As the values of the `XmNxrtTblCellValueContext` and `XmNxrtTblLabelContext` resources
- In the lists used to set the `XmNxrtTblColumnLabels` and `XmNxrtTblRowLabels` resources
- In the callbacks specified by the `XmNxrtTblCellValueCallback` and `XmNxrtTblLabelCreateCallback` resources
- In the arrays passed to The *XrtTblAddColumns()* and *XrtTblAddRows()*.

If a String is specified, XRT/table automatically converts it to an XmString for its internal use.

The `XmNxrtTblXmStringSetMode` resource enables you to override this default behavior. Resources and procedures that normally expect XmStrings but accept Strings can be set to accept only XmStrings or accept only Strings.

The `XmNxrtTblXmStringGetMode` resource controls the format in which the values of the above resources are retrieved. Normally, the retrieved values are XmStrings, but can be set to be Strings.

2.8 Colors

You can specify colors for the foreground and background of cells, labels, and selected cells. If no colors are specified, XRT/table inherits the default colors from the parent's `XmNforeground` and `XmNbackground`.

Color Specification

Colors can be specified using either a Pixel color value or a string. Strings contain either the color name or the hexadecimal representation of the color's Red, Green and Blue (RGB) components. For example:

```
char * my_named_color = "LemonChiffon";
char * my_rgb_color = "#228B22";
```

If a color is specified by name, the name should appear in the X Server's color database. On Unix systems, this is normally the file */usr/lib/X11/rgb.txt*.

If the color's RGB value is specified instead of a named color, the string must begin with a '#' character followed by a two-character hexadecimal value for each of the Red, Green and Blue components. In the example above, `my_rgb_color` is composed of `0x22` units of Red, `0x8B` units of Green and `0x22` units of Blue.[1]

1. The syntax for color specification is not quite as restrictive as described above. Any string acceptable to the Xlib procedure *XParseColor()* may be specified.

When an application built with XRT/table is displayed on a monochrome X Server:

- All non-black colors appear as white when used for background colors.
- All other non-white colors appear as black.

Once an XRT/table widget is created, an application should not update or change the application's colormap. Changes to any of the colors should be made by updating the appropriate widget resources (such as `XmNxrtTblBackgroundContext`).

Colormap Flashing

X Server colormaps are a shared resource. In some situations, the number of colors in use by all of the applications displayed is more than the number of colormap entries available. In this case, the colors in some windows will "flash" to inappropriate colors as the user moves the mouse around the screen.

One way to minimize color flashing is for the Server to provide only *static* colors. In this case, applications will not be given the exact colors they ask for, but will instead be given the closest color available within the window's colormap.

However, many applications require dynamic colors. In this case, each application will be given exactly the colors it asks for[1]. Depending on the number of colors the Server has available and the number and state of running applications, colormap flashing may occur.

When running on a Server with a dynamic colormap, XRT/table applications will automatically allocate and de-allocate color cells in the Server as required by the application. This is done to minimize demands on the colormap resource by the XRT/table application. If an XRT/table application requests more colors than are available, black or white will be used instead of the unavailable color.

If your table cells or labels contain pixmaps, ensure that your pixmaps do not use more colors than the user's Server can display.

2.9 Warnings and Error Messages

Setting `XmNxrtTblDebug` to `True` will cause XRT/table to print warning and error messages to *stderr* from the Xt Error Database when it discovers any inconsistencies. This will occur, for example, if you set the number of frozen rows to be greater than the number of rows in the table.

1. Within the capabilities of the display's hardware.

2.10 Using Resource Files

Sometimes it is better to use resource files instead of C code to set the value of resources. Resources files provide the following benefits:

- Speed – Changes can be seen immediately, instead of having to be compiled.
- Flexibility – You can allow end-users to modify some resource values to suit their own needs.

Most XRT/table resources can be set in resource files, including cell values. However, most resources are too complex or too obscure for end-users to want to modify them. Resource files for all the widgets in an application can be combined to create an application defaults file.

For further information on resource files and their usage, consult your Xt Intrinsics documentation. Appendix G on page 257 provides details on the syntax of resource settings when used in resource files, as well as a sample table resource file.

3

Programming XRT/table

Resource Summary ■ *Number of Rows/Columns* ■ *Visible Rows/Columns*
Frozen Rows and Columns ■ *Row and Column Labels* ■ *Table Titles*
Row Height and Column Width ■ *Cell and Label Margins*
Foreground, Background and Shadow Colors ■ *Cell and Label Text Alignment*
Cell and Label Font ■ *Border Types and Sides* ■ *Background Pixmap*
Text Clipping and Overflow ■ *Cell and Label Spanning*
User Data ■ *Table Printing* ■ *Cell and Label Flashing*

3.1 Resource Summary

The following figures summarize all of the XRT/table resources. It is not necessary to remember all the resources to program XRT/table effectively. For most tables many resources may be left at their default settings.

Resource	Type
XmNxrtTblAllowCellResize	XrtTblResize
XmNxrtTblAllowResize	XrtTblResize
XmNxrtTblCellValueFunc	XrtTblXmStringProc
XmNxrtTblCellValues	XrtTblCellValues *
XmNxrtTblColumnLabelOffset	int
XmNxrtTblCursorCorner	Cursor
XmNxrtTblCursorHorizontal	Cursor
XmNxrtTblCursorNontraversable	Cursor
XmNxrtTblCursorTraversable	Cursor
XmNxrtTblCursorVertical	Cursor
XmNxrtTblDataTypeList	XrtTblDataTypeRoutine *
XmNxrtTblDisplayClipArrows	Boolean
XmNxrtTblDoubleBuffer	Boolean

Continued on next page...

Resource	Type
XmNxrtTblExportDelimiter	String
XmNxrtTblExportTargets	XrtTblImportExportStruct **
XmNxrtTblFrameBorderType	XrtTblBorderType
XmNxrtTblFrameShadowThickness	Dimension
XmNxrtTblFrozenColumnPlacement	XrtTblColumnPlacement
XmNxrtTblFrozenColumns	int
XmNxrtTblFrozenRowPlacement	XrtTblRowPlacement
XmNxrtTblFrozenRows	int
XmNxrtTblImportDelimiter	String
XmNxrtTblImportTargets	XrtTblImportExportStruct **
XmNxrtTblLeftColumn	int
XmNxrtTblMarginHeight	Dimension
XmNxrtTblMarginWidth	Dimension
XmNxrtTblMinCellVisibility	int
XmNxrtTblMinFlexibleHeight	Dimension
XmNxrtTblMinFlexibleWidth	Dimension
XmNxrtTblMode	XrtTblMode
XmNxrtTblNumColumns	int
XmNxrtTblNumRows	int
XmNxrtTblPSFontMapList	XrtTblPSFontMap *
XmNxrtTblRepeatBackgroundColors	String *
XmNxrtTblRepeatForegroundColors	String *
XmNxrtTblRowLabelOffset	int
XmNxrtTblSelectedBackground	String
XmNxrtTblSelectedCellList	XrtTblRange **
XmNxrtTblSelectedForeground	String
XmNxrtTblSelectionPolicy	XrtTblSelectionPolicy
XmNxrtTblSetTextCursorPos	Boolean
XmNxrtTblShadowThickness	Dimension
XmNxrtTblSpanList	XrtTblRange **
XmNxrtTblTextInitSelect	Boolean
XmNxrtTblTopRow	int
XmNxrtTblTrackCursor	Boolean
XmNxrtTblValidatePolicy	XrtTblValidatePolicy
XmNxrtTblVisibleColumns	int
XmNxrtTblVisibleRows	int
XmNxrtTblWidgetClassList	XrtTblWidgetClass *
XmNxrtTblXmStringGetMode	XrtTblXmStringGet
XmNxrtTblXmStringSetMode	XrtTblXmStringSet

Figure 11 Table-wide Resources

Resource	Type
XmNxrtTblAlignmentContext	XrtTblAlignment
XmNxrtTblArgListContext	ArgList
XmNxrtTblBackgroundContext	String
XmNxrtTblBackgroundPixelContext	Pixel
XmNxrtTblBorderSidesContext	XrtTblBorderSides
XmNxrtTblBorderTypeContext	XrtTblBorderType
XmNxrtTblBottomShadowContext	Pixel
XmNxrtTblCharHeightContext	int
XmNxrtTblCharWidthContext	int
XmNxrtTblClipPixmapContext	Pixmap
XmNxrtTblContext	XrtTblContext *
XmNxrtTblDataTypeContext	XrtTblDataType
XmNxrtTblFieldArgListContext	ArgList
XmNxrtTblFontListContext	XmFontList
XmNxrtTblForegroundContext	String
XmNxrtTblForegroundPixelContext	Pixel
XmNxrtTblOverflowContext	XrtTblOverflow
XmNxrtTblPixelHeightContext	int
XmNxrtTblPixelWidthContext	int
XmNxrtTblPixmapContext	Pixmap
XmNxrtTblPixmapLayoutContext	XrtTblPixmapLayout
XmNxrtTblPixmapShadowContext	Boolean
XmNxrtTblPSFontListContext	String
XmNxrtTblRangeContext	XrtTblRange *
XmNxrtTblSensitiveContext	Boolean
XmNxrtTblTopShadowContext	Pixel
XmNxrtTblUserDataContext	XtPointer
XmNxrtTblWidgetContext	Widget
XmNxrtTblWidgetShadowContext	Boolean
XmNxrtTblWordWrapContext	Boolean

Figure 12 Cell/Label Context Resources

Resource	Type
XmNxrtTblCaseContext	XrtTblCase
XmNxrtTblCellValueContext	XmString
XmNxrtTblCurrencyPickListContext	double *
XmNxrtTblDatePickListContext	struct tm **
XmNxrtTblEditableContext	Boolean
XmNxrtTblFloatPickListContext	double *
XmNxrtTblIntPickListContext	long *
XmNxrtTblMaxLengthContext	int
XmNxrtTblMultilineContext	Boolean
XmNxrtTblStringPickListContext	String
XmNxrtTblTraversableContext	Boolean

Figure 13 Cell-only Context Resources

Resource	Type
XmNxrtTblColumnLabelPlacement	XrtTblRowPlacement
XmNxrtTblColumnLabels	XrtTblXmStringList *
XmNxrtTblLabelContext	XmString
XmNxrtTblLabelCreateFunc	XrtTblXmStringProc
XmNxrtTblRowLabelPlacement	XrtTblColumnPlacement
XmNxrtTblRowLabels	XrtTblXmStringList *

Figure 14 Label-only Resources

Resource	Type
XmNxrtTblAlignmentSeries	XrtTblSeries *
XmNxrtTblArgListSeries	XrtTblSeries *
XmNxrtTblBackgroundSeries	XrtTblSeries *
XmNxrtTblBorderSidesSeries	XrtTblSeries *
XmNxrtTblBorderTypeSeries	XrtTblSeries *
XmNxrtTblCharHeightSeries	XrtTblSeries *
XmNxrtTblCharWidthSeries	XrtTblSeries *
XmNxrtTblClipPixmapSeries	XrtTblSeries *
XmNxrtTblDataTypeSeries	XrtTblSeries *
XmNxrtTblFieldArgListSeries	XrtTblSeries *
XmNxrtTblFontListSeries	XrtTblSeries *
XmNxrtTblForegroundSeries	XrtTblSeries *
XmNxrtTblOverflowSeries	XrtTblSeries *
XmNxrtTblPixelHeightSeries	XrtTblSeries *
XmNxrtTblPixelWidthSeries	XrtTblSeries *
XmNxrtTblPixmapSeries	XrtTblSeries *
XmNxrtTblPixmapLayoutSeries	XrtTblSeries *
XmNxrtTblPixmapShadowSeries	XrtTblSeries *
XmNxrtTblPSFontListSeries	XrtTblSeries *
XmNxrtTblSensitiveSeries	XrtTblSeries *
XmNxrtTblUserDataSeries	XrtTblSeries *
XmNxrtTblWidgetSeries	XrtTblSeries *
XmNxrtTblWidgetShadowSeries	XrtTblSeries *
XmNxrtTblWordWrapSeries	XrtTblSeries *

Figure 15 Cell/Label Series Resources

Resource	Type
XmNxrtTblCaseSeries	XrtTblSeries *
XmNxrtTblCurrencyPickListSeries	XrtTblSeries *
XmNxrtTblDatePickListSeries	XrtTblSeries *
XmNxrtTblEditableSeries	XrtTblSeries *
XmNxrtTblFloatPickListSeries	XrtTblSeries *
XmNxrtTblIntPickListSeries	XrtTblSeries *
XmNxrtTblMaxLengthSeries	XrtTblSeries *
XmNxrtTblMultilineSeries	XrtTblSeries *
XmNxrtTblStringPickListSeries	XrtTblSeries *
XmNxrtTblTraversableSeries	XrtTblSeries *

Figure 16 Cell-only Series Resources

Resource	Type
XmNxrtTblDisplayHorizScrollBar	XrtTblDispScrollBar
XmNxrtTblDisplayVertScrollBar	XrtTblDispScrollBar
XmNxrtTblHorizScrollBarLeftAttachment	XrtTblAttachment
XmNxrtTblHorizScrollBarPlacement	XrtTblRowPlacement
XmNxrtTblHorizScrollBarPosition	XrtTblSBPosition
XmNxrtTblHorizScrollBarRightAttachment	XrtTblAttachment
XmNxrtTblJumpScroll	XrtTblJumpScroll
XmNxrtTblPreviewScrolling	XrtTblPreviewScroll
XmNxrtTblSpace	Dimension
XmNxrtTblVertScrollBarBottomAttachment	XrtTblAttachment
XmNxrtTblVertScrollBarPlacement	XrtTblColumnPlacement
XmNxrtTblVertScrollBarPosition	XrtTblSBPosition
XmNxrtTblVertScrollBarTopAttachment	XrtTblAttachment

Figure 17 Scrollbar Resources

Resource	Type
XmNxrtTblClipChildren	Widget *
XmNxrtTblClipChildrenPersistent	Boolean
XmNxrtTblHorizontalScrollBar	Widget
XmNxrtTblText	Widget
XmNxrtTblVerticalScrollBar	Widget

Figure 18 Child Widget Resources

Resource	Type
XmNxrtTblBottomOffset	int
XmNxrtTblLeftOffset	int
XmNxrtTblLocation	XrtTblLocation
XmNxrtTblResizeToCell	Boolean
XmNxrtTblRightOffset	int
XmNxrtTblTopOffset	int
XmNxrtTblWidgetLocation	XrtTblContext *

Figure 19 Constraint Resources

Resource	Type
XmNxrtTblCellValueCallback	XtCallbackList
XmNxrtTblContextValueCallback	XtCallbackList
XmNxrtTblCreateWidgetCallback	XtCallbackList
XmNxrtTblDisplayWidgetCallback	XtCallbackList
XmNxrtTblEnterCellCallback	XtCallbackList
XmNxrtTblExportCallback	XtCallbackList
XmNxrtTblExposeCallback	XtCallbackList
XmNxrtTblImportCallback	XtCallbackList
XmNxrtTblLabelCreateCallback	XtCallbackList
XmNxrtTblPrintCallback	XtCallbackList
XmNxrtTblPrintWidgetCallback	XtCallbackList
XmNxrtTblResizeCallback	XtCallbackList
XmNxrtTblResizeCellCallback	XtCallbackList
XmNxrtTblScrollCallback	XtCallbackList
XmNxrtTblSelectCallback	XtCallbackList
XmNxrtTblSortCallback	XtCallbackList
XmNxrtTblTrackCursorCallback	XtCallbackList
XmNxrtTblTraverseCellCallback	XtCallbackList
XmNxrtTblValidateCellCallback	XtCallbackList
XmNxrtTblWidgetSetValueCallback	XtCallbackList

Figure 20 Callback Resources

Resource	Type
XmNxrtTblDebug	Boolean
XmNxrtTblRepaint	XrtTblRepaint

Figure 21 Other Resources

3.2 Number of Rows/Columns

The `XmNxrtTblNumRows` and `XmNxrtTblNumColumns` resources set the number of rows and columns in the table. They must be set for most tables because they define the size of the table shown to the user.

These resources are independent of the `XrtTblCellValues` structure that stores the cell values. `XmNxrtTblNumRows`/`XmNxrtTblNumColumns` can be smaller or larger than the number of rows/columns in the table's cell values structure, as shown in Figure 22.

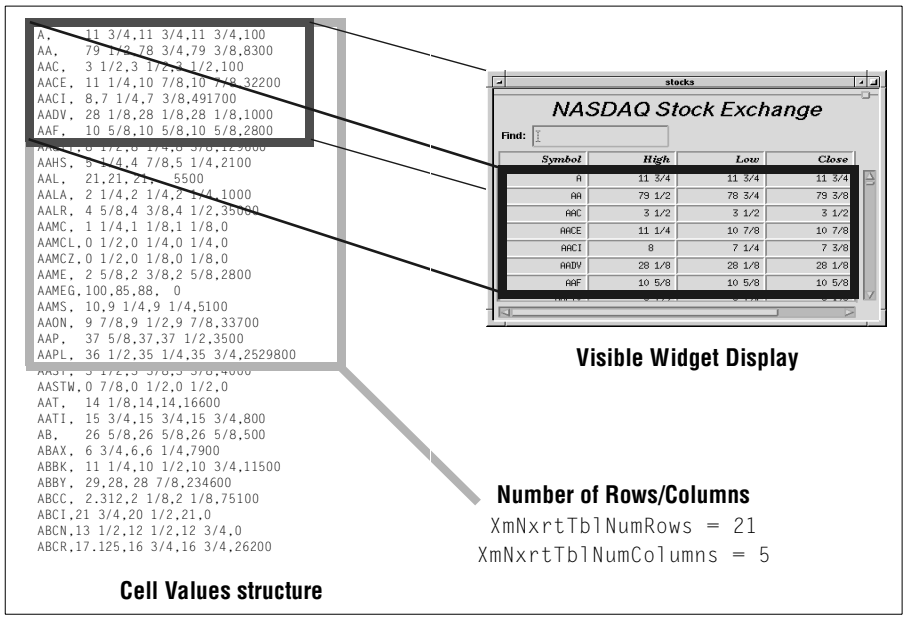


Figure 22 Using XmNxrtTblNumRows to limit number of scrollable rows

For example, you could temporarily "delete" rows from the bottom of the table by reducing the value of `XmNxrtTblNumRows`. The cell values of the "deleted" rows are still in the `XrtTblCellValues` structure. The rows could be restored later by increasing `XmNxrtTblNumRows`.

The number of rows/columns cannot be less than or equal to the number of frozen rows/columns.

3.3 Visible Rows/Columns

The number of rows and columns currently visible in the window is specified by `XmNxrtTblVisibleRows` and `XmNxrtTblVisibleColumns`.[1]

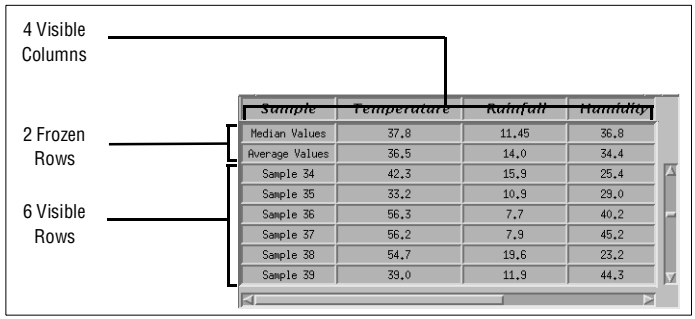


Figure 23 Visible and Frozen Rows and Columns

These resources are dynamically updated as other table resources change and when a user interacts with the widget. The value does not include any frozen rows/columns, as Figure 23 illustrates.

You can force the table to display a particular number of rows or columns by setting these resources. Setting visible rows/columns always causes the table to attempt to resize itself, overriding the resize policy set by `XmNxrtTblAllowResize`.

You cannot set the number of visible rows or columns to greater than the corresponding `XmNxrtTblNumRows` and `XmNxrtTblNumColumns` resources.

Displaying Entire Table

To display the entire table, set `XmNxrtTblVisibleRows` and `XmNxrtTblVisibleColumns` to 0. Setting either resource to zero sets a special flag that causes it to attempt to resize to display all rows/columns in the table. Any change to the number of rows/columns in the table will cause the table to attempt to resize when this flag has been set.

3.4 Frozen Rows and Columns

An application can make rows and columns non-scrollable using `XmNxrtTblFrozenRows` and `XmNxrtTblFrozenColumns`. `XmNxrtTblFrozenRows` specifies the number of rows held at the top or bottom of the window and not scrolled. `XmNxrtTblFrozenColumns` specifies the number of columns held at the left or right side of the window and not scrolled.

1. Rows/columns that are only partially visible are also included in the value of these resources.

Frozen rows/columns always start from the beginning of the table. They are still editable and traversable unless set otherwise. Figure 23 shows an example of frozen rows.

Use frozen rows/columns to hold important information (such as totals at the bottom of the table) on the screen as a user scrolls through the table. Another use is as additional rows/columns that act like labels.

Freezing Rows and Columns

To freeze rows/columns, call *XrtTblMoveRows()* or *XrtTblMoveColumns()* to move the desired rows/columns to the beginning of the table. Then set `XmNxrtTblFrozenRows` or `XmNxrtTblFrozenColumns` to the number of rows/columns from the beginning that you want to freeze.

For example, the following code moves row 5 to the beginning of the table, and then freezes it:

```
XrtTblMoveRows(table, 5, 1, 0, True);
XtVaSetValues(table,
     XmNxrtTblFrozenRows,       1,
     XmNxrtTblFrozenColumns,    0,
     NULL);
```

Frozen Row/Column Placement

You can place frozen rows at either the top or bottom of the widget display. Frozen columns can be placed at either the left or right of the widget display. The placement of frozen rows/columns does not affect the location of the rows/columns in the `XrtTblCellValues` structure.

To change the placement of all frozen rows, set `XmNxrtTblFrozenRowPlacement` to either `XRTTBL_PLACE_TOP` or `XRTTBL_PLACE_BOTTOM`.

To change the placement of all frozen columns, set `XmNxrtTblFrozenColumnPlacement` to either `XRTTBL_PLACE_LEFT` or `XRTTBL_PLACE_RIGHT`.

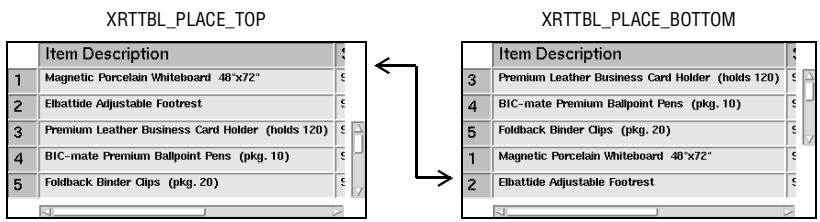


Figure 24 Frozen Rows at top and bottom of window

3.5 Row and Column Labels

The `XrtTblXmStringList` structure pointed to by `XmNxrtTblColumnLabels` specifies the column labels. The `XrtTblXmStringList` structure pointed to by

XmNxrtTblRowLabels specifies the row labels. Row/column labels are stored separately from the cell values.

Users cannot interactively edit labels.

3.5.1 Setting and Retrieving Label Values

Setting All Labels

To specify all of the labels for a row or column, set XmNxrtTblColumnLabels or XmNxrtTblRowLabels to a new list of labels, as shown by the following example:

```
XrtTblXmStringList labels;
String label_list[] = { "Row 0", "Row 1", "Row 2" };

labels.length = XtNumber(label_list);
labels.list = (XmString *)label_list;
XtVaSetValues(table,
    XmNxrtTblRowLabels,         &labels,
    NULL);
```

By default, XmNxrtTblColumnLabels and XmNxrtTblRowLabels can accept either a list of XmStrings or a list of Strings. For information on how to restrict the list to one type or the other by setting the XmNxrtTblXmStringSetMode resource, see section 4.3.1 on page 77.[1]

Setting Individual Labels

To set individual labels, set a context to one or more labels and then set the XmNxrtTblLabelContext resource, as shown by the following:

```
String labels[] = { "Name", "Address", "Phone Number",
                    "Position", "Salary", NULL };
for (i=0; labels[i]; i++)
    XtVaSetValues(table,
        XmNxrtTblContext,  XrtTblSetContext(XRTTBL_LABEL,
                                i),
        XmNxrtTblLabelContext, labels[i],
        NULL);
```

Retrieving a Label

There are two ways to retrieve label values:

1. Use XmNxrtTblLabelContext to get the value of a label when the context is a single label, for example:

```
XmString value1;
XtVaSetValues(table,
    XmNxrtTblContext, XrtTblSetContext(XRTTBL_LABEL,0),
    NULL);
XtVaGetValues(table,
    XmNxrtTblLabelContext, &value1,
    NULL);
value1 = XmStringCopy(value1);
```

1. If you are linking to Motif 2.x, XmNxrtTblColumnLabels and XmNxrtTblRowLabels cannot accept both XmString and String lists. You must use XmNxrtTblXmStringSetMode to specify which type of list you want to use.

2. Use the LABEL_VALUE macro to get the value of a row or column label at a position.

Specifying the Label Type

The XmNxrtTblXmStringGetMode resource allows you to specify whether values retrieved from cells or labels should be of type String or type XmString. The XmNxrtTblXmStringSetMode resource allows you to specify whether labels being set by the application must be Strings or XmStrings. These resources are described in section 4.3.1 on page 77.

3.5.2 Label Placement and Spacing

Label Placement

You can specify the positioning of row/column labels on the screen using XmNxrtTblColumnLabelPlacement and XmNxrtTblRowLabelPlacement.

Column labels can be displayed along the top (XRTTBL_PLACE_TOP) or bottom (XRTTBL_PLACE_BOTTOM) side of the table. Row labels can be displayed along the left (XRTTBL_PLACE_LEFT) or right (XRTTBL_PLACE_RIGHT) side of the table.

Magnetic Porcelain Whiteboard 48"x72"	940-874	100	1
Elbattide Adjustable Footrest	940-538	50	2
Premium Leather Business Card Holder (holds 120)	940-249	250	3
BIC-mate Premium Ballpoint Pens (pkg. 10)	940-859	1000	4
Foldback Binder Clips (pkg. 20)	940-903	2500	5
Item Description	SKU #	Qty	

Figure 25 Reversing the default row/column label placement

Label Spacing

Normally, there is no space between labels and the cell area. The XmNxrtTblRowLabelOffset resource allows you to specify the distance in pixels between the row labels and the cell area, and the XmNxrtTblColumnLabelOffset resource specifies the distance in pixels between the column labels and the cell area. If a negative value is specified, the labels overlap the cell area.

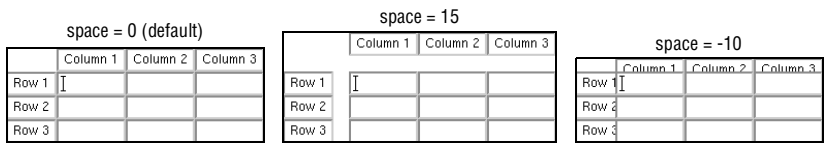


Figure 26 Row and column label spacing

3.5.3 Label Creation Callbacks

Another way to set label values is using a LabelCreate callback. This requires more work in the application but improves the performance of large tables.

The `XmNxrtTblLabelCreateCallback` resource specifies a callback routine that XRT/table calls at runtime when it first displays any blank label.[1] The application is completely responsible for performing any addition, deletion, moving, or sorting of the label values that may be required.

The callback you create can be set to allocate the returned XmString (XRT/table frees it) or pass a static pointer, and is passed the following structure:

```
typedef struct {
    int         reason;         // Read-only
    XEvent      *event;         // Read-only
    int         row, column;    // Read-only
    XmString    string;
    Boolean     store;
    Boolean     free_string;
} XrtTblLabelCreateCallbackStruct;
```

By default, *string* can be either an XmString or a String[2]. If you want to force *string* to be of one type or the other, set the `XmNxrtTblXmStringSetMode` resource.

The following example lists a simple LabelCreate callback:

```
static XmString
labelcreateCB(Widget table, XtPointer data,
    XrtTblLabelCreateCallbackStruct *cbs)
{
    static char buf[20];

    if (IS_ROWLABEL(cbs->row, cbs->column))
        /* Row Label */
        sprintf(buf, "%d", cbs->row+1);
    else    /* Column Label */
        sprintf(buf, "%d", cbs->column+1);
    cbs->string = buf;
    cbs->free_string = False;
}
```

This example uses a static buffer to store the label value, and sets the *free_string* member of the `XrtTblLabelCreateCallbackStruct` to tell XRT/table not to free the label when finished with it.

To add this callback to the LabelCreate callback list, call *XrtTblAddCallback()*:

```
XrtTblAddCallback(table, XmNxrtTblLabelCreateCallback,
    labelcreateCB, NULL);
```

Calling *XrtTblAddCallback()* is required to ensure that XRT/table knows of the existence of this callback.

---

1. A label is blank if its `XrtTblXmStringList` element has a `NULL` value.

2. If you are linking with Motif 2.x, *string* cannot accept both XmStrings and Strings. You must use `XmNxrtTblXmStringSetMode` to specify one type or the other.

You can specify whether or not the string is to be stored internally by modifying the *store* member of the label creation structure. If *store* is set to `True`, the value is stored and the callback is not called again. By default, *store* is `False`, and the value is not stored.

XRT/table includes a method that generates spreadsheet-style row and column labels, *XrtTblCreateSpreadsheetLabel()*.

3.6 Table Titles

An application can add one or more titles to any side of a table. A title is a widget that is outside of the table's cells and labels. Once added to the table, XRT/table controls the title's size and position using its constraint resources.

Although any widget can be used as a title, XRT/table includes a new label widget designed to add attractive titles to tables. The XrtLabel widget is similar to the standard Motif XmLabel widget, but adds a shadow border, the ability to rotate its contents, and PostScript printing (when used as a title). See Appendix I on page 271 for complete information on XrtLabel.

You can use titles in a number of other ways, for example as a table header/ footer or as a text area to annotate a particular cell.

Adding a Title

To add a title to a table, create the widget as a child of the table. The following example creates an XrtLabel widget as a table title:

```
title = XtVaCreateWidget("title",
    xtXrtLabelWidgetClass,  table,
    XmNlabelString,         "A Table Title",
    XmNxrtTblLocation,      XRTTBL_LOCATION_TOP,
    NULL);
```

The title widget should be created unmanaged, and management of it left to the table widget.

Positioning a Title

XRT/table's constraint resources control the location (and possibly the size) of each title. Constraint resources must be set on the title widget itself, not on the table.

`XmNxrtTblLocation` specifies which side of the table to place the title. Figure 27 shows most of the valid title locations; if set to `XRTTBL_LOCATION_NONE`, the widget is not displayed. To use a manager widget as a title, add all child widgets to it before setting its `XmNxrtTblLocation` resource.

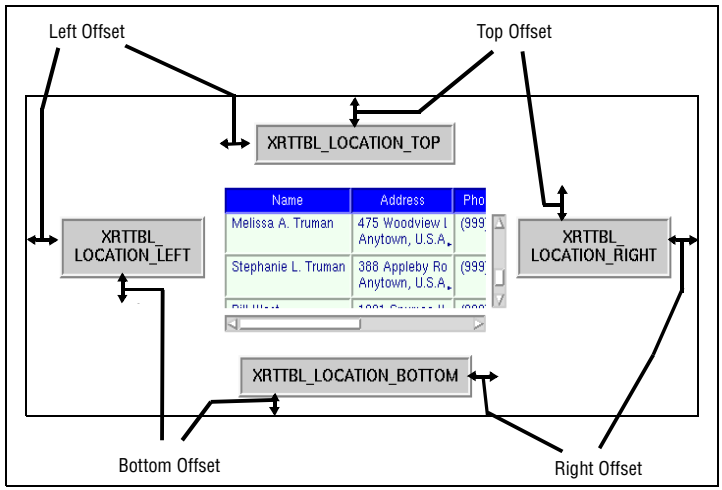


Figure 27 Title locations and offsets

You can adjust the spacing between the edges of the title and the table by using four offset resources: `XmNxrtTblTopOffset`, `XmNxrtTblBottomOffset`, `XmNxrtTblLeftOffset`, and `XmNxrtTblRightOffset`. These resources specify the space around the title. Figure 27 shows how the offsets are measured when the title is on each side of the table. Offsets can be positive, negative, or `XRTTBL_NOVALUE`.

Positive values add space between the title and the table. Negative values remove space between the title and the table. Setting an offset to 0 causes the title to touch the cells/labels or the edge of the table widget.

`XRTTBL_NOVALUE` specifies a floating, unanchored offset; the actual positioning depends on the setting of the opposite offset resource. For example, if `XmNxrtTblLeftOffset` and `XmNxrtTblRightOffset` are set to `XRTTBL_NOVALUE`, XRT/table centers the title horizontally. If the opposite offset resource is set to an explicit pixel value, the title is anchored at that value. Figure 28 shows the effects you can achieve by setting different offset combinations.

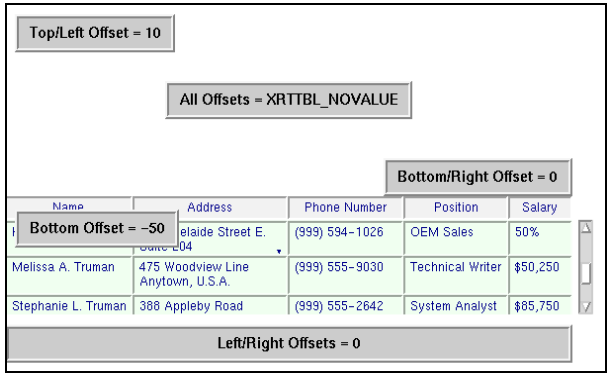


Figure 28 Offset combinations for titles

Rotated Titles

One of the benefits of using the XrtLabel widget for a title is that its text and pixmap can be rotated. To rotate an XrtLabel's text, set `XmNxrtLblRotation` to either `XRTLBL_ROTATE_90` or `XRTLBL_ROTATE_270`.

Resize Handling

During initial display, and anytime the table is resized, XRT/table first calculates the space required to display the titles, using whatever is left for the cells and labels. If the table is made very small, unpredictable results may occur.

Absolute Title Location

An application can determine the coordinates of the top-left corner of the title (relative to the table) by examining the `XmNx` and `XmNy` resources.

Removing a Title

To remove a title from the display, set `XmNxrtTblLocation` to `XRTTBL_LOCATION_NONE`. To permanently destroy a title, call *XtDestroyWidget()*.

3.7 Row Height and Column Width

By default, XRT/table sets the height of rows to display one line of text. The width of columns is set by default to display 10 characters of text. If a cell value, pixmap, or widget does not fit in its cell, the cell displays clipping arrows by default. Each row can have its own height, and each column its own width.

XRT/table provides two different ways to specify row height and column width: *character* and *pixel*. *Character specification* determines the height/width by the number of characters or lines that the row/column can display. *Pixel specification* determines the height/width by the explicit number of pixels.

Only one method can be used for a row or column. Pixel specification overrides character specification.

Note: When users interactively resize rows/columns, the row height/column width is specified by pixel regardless of how your application specified it. You can be notified of user resizing by using a ResizeCell callback.

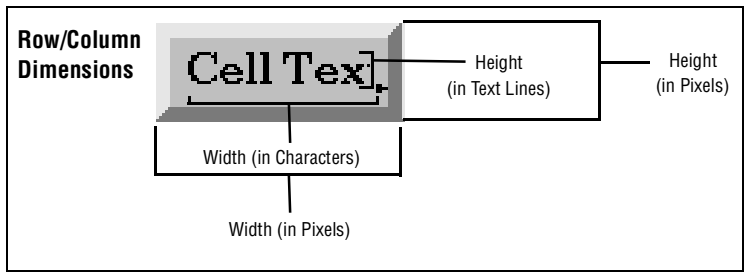


Figure 29 Difference between Character and Pixel Row/Column specification

3.7.1 Character Height and Width

`XmNxrtTblCharWidthContext` specifies the number of characters a column can display. `XmNxrtTblCharHeightContext` specifies the number of lines of text a row can display. For these resources to control row height/column width, `XmNxrtTblPixelWidthContext` and `XmNxrtTblPixelHeightContext` must be `XRTTBL_NOVALUE`.

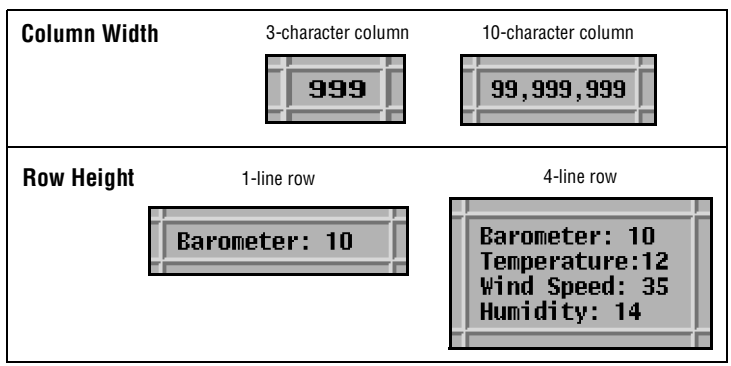


Figure 30 Character specification of row height

Character specification is convenient when you know how many characters you want a row/column to be able to display. It works best with *non-proportional*[1] fonts because XRT/table uses the widest character along with the largest

1. All of the characters in a non-proportional (fixed-width) font have the same width and height.

ascender/descender to guarantee that the specified number of characters will fit in the cell or label.

The following example sets the width of column 10 to 15 characters:

```
XtVaSetValues(table,
    XmNxrtTblContext,  XrtTblSetContext(XRTTBL_ALL, 10),
    XmNxrtTblPixelWidthContext, XRTTBL_NOVALUE,
    XmNxrtTblCharWidthContext,  15,
    NULL);
```

To determine the pixel dimensions of a row or column whose height/width was set by character, use *XrtTblGetCellDimensions()*.

3.7.2 Pixel Height and Width

`XmNxrtTblPixelWidthContext` and `XmNxrtTblPixelHeightContext` specify column width and row height in pixels. You can set these resources to an explicit pixel value, `XRTTBL_NOVALUE`, `XRTTBL_VARIABLE`, or `XRTTBL_FLEXIBLE`. (These last two values are discussed in detail in the following sections.)

Unless set to `XRTTBL_NOVALUE` (the default), these resources control the height and width instead of the character specification resources. Figure 31 shows setting `XmNxrtTblPixelHeightContext` to a pixel value.

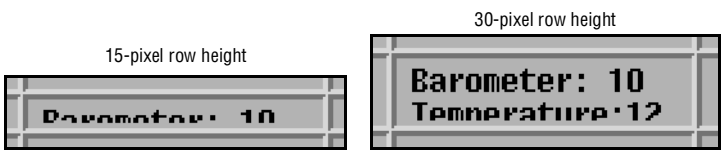


Figure 31 Pixel specification of row height

The following code sets the width of column 10 to 50 pixels:

```
XtVaSetValues(table,
    XmNxrtTblContext,           XrtTblSetContext(XRTTBL_ALL, 10),
    XmNxrtTblPixelWidthContext, 50,
    NULL);
```

3.7.3 Variable Height and Width

An application can have XRT/table automatically size rows and columns to fit the contents of the table by setting `XmNxrtTblPixelWidthContext` and `XmNxrtTblPixelHeightContext` to `XRTTBL_VARIABLE`. As your application changes table attributes that affect the cells' contents, XRT/table will keep resizing the rows and columns to fit.[1]

1. When there are no values in a variable height row (or a variable width column) the row/column becomes hidden.

When a cell contains a cell value and a background pixmap, XRT/table sizes the cell to fit the larger of the two. When a cell contains a widget, XRT/table sizes the cell to fit the widget's preferred size (determined by the *XtQueryGeometry()* procedure).

Note: Once a user has resized a row or column, variable row/column resizing ends because the row/column size is specified by an explicit number of pixels. User cell editing does not automatically change the size of rows/columns.

If your application is making a lot of changes to a table, turn repaint off by setting `XmNxrtTblRepaint` to `XRTTBL_REPAINT_OFF`, make the changes, and then turn repaint on to update the table display.

To determine the pixel dimensions of a row or column with variable height or width, use *XrtTblGetCellDimensions()*.

3.7.4 Flexible Rows and Columns

An application can specify that certain rows or columns are flexible by setting `XmNxrtTblPixelHeightContext` or `XmNxrtTblPixelWidthContext` to `XRTTBL_FLEXIBLE`. When set, resizing the table causes these rows/columns to resize as well, as shown by Figure 32.

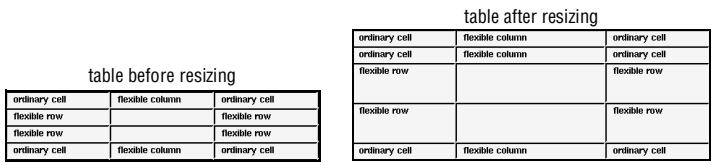


Figure 32 Resizing a table containing flexible rows and columns

When flexible rows/columns are resized, they are resized to equal heights/widths.

To set a minimum height and width for all flexible row/columns, use The `XmNxrtTblMinFlexibleHeight` and `XmNxrtTblMinFlexibleWidth` resources. For example, the following code sets a minimum flexible row height of 20 and a minimum flexible column width of 50:

```
XtVaSetValues(table,
    XmNxrtTblMinFlexibleHeight, 20,
    XmNxrtTblMinFlexibleWidth,  50,
    NULL);
```

3.7.5 Hiding Rows and Columns

An application can "hide" rows and columns from the end-user by setting `XmNxrtTblPixelHeightContext/XmNxrtTblPixelWidthContext` to 0 pixels.[1] Though the row/column appears to have vanished, the application can set attributes or change cell values. Figure 33 illustrates the effect.

1. Users cannot interactively hide or reveal rows/columns.

The current cell should not be in the hidden row/column.

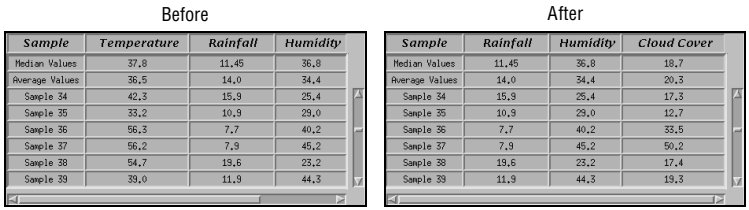


Figure 33 Hiding the "Temperature" column

To reveal a hidden row/column, set the pixel height or width to a pixel value or `XRTTBL_NOVALUE` (to use the character specification defined for the row/column).

3.8 Cell and Label Margins

You can alter the space between the cell borders and the contents of cells using `XmNxrtTblMarginHeight` and `XmNxrtTblMarginWidth`. Figure 34 shows the appearance of different margins. These resources affect all cells/labels in the table—margins cannot be set for individual cells.

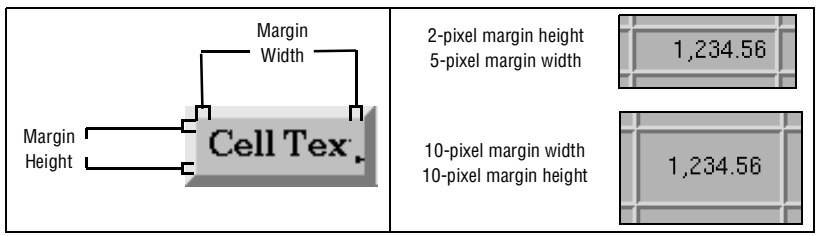


Figure 34 Margin Height and Margin Width

The margin resources cannot be smaller than the value of `XmNhighlightThickness`, because the current cell's XmText widget draws its highlight in the margin space.

3.9 Foreground, Background and Shadow Colors

Specifying Colors

XRT/table's foreground/background color resources take a color string (as described in section 2.8 on page 29) as their value. Foreground and background colors should be complementary, with enough contrast between them to make the text easy to read. Pastel colors such as "LightBlue" or "Wheat" often make good background colors. Saturated colors are often good foreground colors. For example:

```
XtVaSetValues(table,
    XmNxrtTblContext,           XrtTblSetContext(5, XRTTBL_ALL),
    XmNxrtTblBackgroundContext, "LightBlue",
    XmNxrtTblForegroundContext, "NavyBlue",
    NULL);
```

Using Pixel Values

If you want to use Pixel values to set XRT/table's foreground and background color resources, set the `XmNxrtTblBackgroundPixelContext` and `XmNxrtTblForegroundPixelContext` resources.

When `XmNxrtTblBackgroundPixelContext` is set, `XmNxrtTblBackgroundContext` is changed (and vice versa). Similarly, setting `XmNxrtTblForegroundPixelContext` changes the value of `XmNxrtTblForegroundContext`.

Background

The table's background color is determined by the inherited `XmNbackground` resource. Each cell and label in the table can have its own background color, specified with `XmNxrtTblBackgroundContext`. By default, the background of all cells/labels is also inherited from `XmNbackground`.

Foreground

The default foreground color for the table is determined by the inherited `XmNforeground` resource. Each cell and label in the table can have its own foreground color, specified with `XmNxrtTblForegroundContext`. By default, the foreground of all cells/labels is also inherited from `XmNforeground`.

Selected Cells

The background/foreground colors used for selected cells are specified by `XmNxrtTblSelectedBackground` and `XmNxrtTblSelectedForeground`. By default, selected cells have a black background and white foreground. The current cell displays the selection colors in its border.

Set `XmNxrtTblSelectedBackground` to `None` to use the foreground and background colors defined for the cells (`XmNxrtTblForegroundContext` and `XmNxrtTblBackgroundContext`). This makes a selection appear "invisible"–however, any `XmNxrtTblSelectCallback` defined is called.

Child Widgets

An application can specify the colors of the child widgets managed by the table (the scrollbars, the XmText current cell, and titles) by setting the widget's `XmNbackground`

and XmNforeground resources. To set colors of widgets in cells/labels, set the XmNxrtTblBackgroundContext and XmNxrtTblForegroundContext resources for the cell/label.

Repeating Colors

XRT/table makes it easy to create rows or columns whose background and foreground colors alternate or cycle in a repeating pattern. To create a repeating pattern of background colors, set the XmNxrtTblRepeatBackgroundColors resource as shown by the following example:

```
String backcolors[] = {"grey95", "grey65", NULL};

XtVaSetValues(table,
    XmNxrtTblContext, XrtTblSetContext(XRTTBL_ALL,XRTTBL_ALL),
    XmNxrtTblBackgroundContext, XRTTBL_REPEAT_ROW,
    XmNxrtTblRepeatBackgroundColors, backcolors,
    NULL);
```

The first two lines specify the context in which to apply the repeating background.

A list of repeating foreground colors can be created by setting the XmNxrtTblRepeatForegroundColors resource.

You can define as many repeating colors as you like. The colors are always selected in the order listed.

To set the background color for the current context to the repeating color list defined by XmNxrtTblRepeatBackgroundColors, set XmNxrtTblBackgroundContext to either XRTTBL_REPEAT_ROW (repeat in row order) or XRTTBL_REPEAT_COLUMN (repeat in column order).

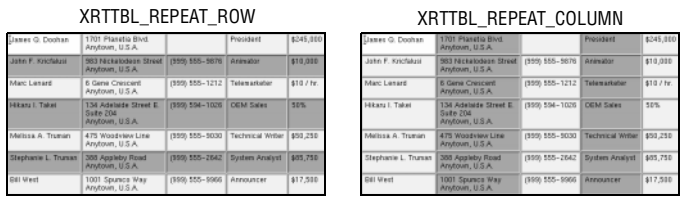


Figure 35 Repeating background colors

To set the foreground colors for the current context to a repeating color list defined by XmNxrtTblRepeatForegroundColors, set XmNxrtTblForegroundContext to XRTTBL_REPEAT_ROW or XRTTBL_REPEAT_COLUMN.

Shadow Colors

The XmNxrtTblBottomShadowContext and XmNxrtTblTopShadowContext resources specify the bottom and top shadow colors for the current context. XRT/table determines appropriate shadow colors automatically when these resources are set to XRTTBL_NOVALUE.

Shadow thickness is controlled by the XmNxrtTblShadowThickness resource.

3.10 Cell and Label Text Alignment

The horizontal and vertical alignment of text within cells and labels is specified by the XmNxrtTblAlignmentContext resource. Cell/label values can be centered or positioned along any side of the cell/label. Figure 36 shows how the values of this resource affect the text display (when XmNstringDirection is XmSTRING_DIRECTION_L_TO_R).

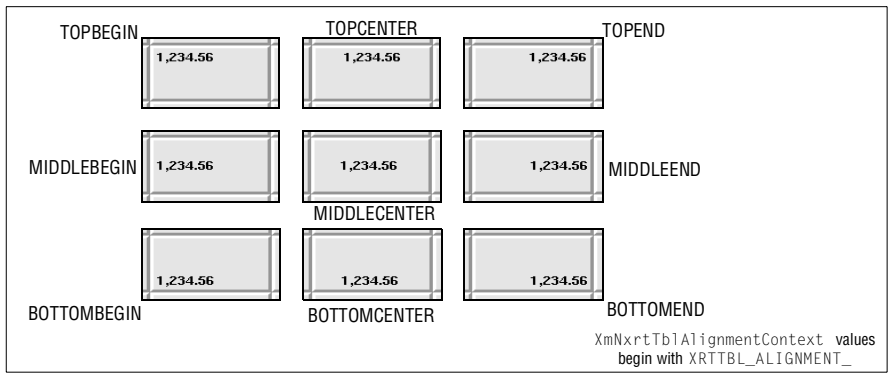


Figure 36 Text alignment positions

The way that text is actually drawn depends on the setting of the inherited XmNstringDirection resource. For example, setting XmNxrtTblAlignmentContext to XRTTBL_ALIGNMENT_TOPBEGIN when XmNstringDirection is set to XmSTRING_DIRECTION_R_TO_L causes text to be drawn from right to left and to appear right-aligned.

3.11 Cell and Label Font

You can specify the font for the text in a cell or label with XmNxrtTblFontListContext. XRT/table supports the use of one or more fonts in each cell/label. For more information on creating and using Compound Strings, see section 6.5 on page 124.

XRT/table can use any X font available on the X Server. If you choose to use a named font, use the X Window program *xlsfonts* or the Xlib call *XListFonts()* to determine which fonts are available on the Server. Use the Xlib call *XLoadFont()* or *XLoadQueryFont()* to obtain the font's resource ID. See your X programming documentation for details on finding and setting fonts.

The easiest way to specify a fontlist is to use the Motif resource converter to convert the font string to the XmFontList data type that `XmNxrtTblFontListContext` requires, as shown below:

```
XtVaSetValues(table,
    XmNxrtTblContext, XrtTblSetContext(XRTTBL_LABEL,XRTTBL_ALL),
      XtVaTypedArg,   XmNxrtTblFontListContext,
      XmRString, "-*-times-medium-*-*--18-*-iso8859-*",
      strlen("-*-times-medium-*-*--18-*-iso8859-*") +1,
    NULL);
```

3.12 Border Types and Sides

All cells and labels have a border around them. The visual look of the border, and the sides on which it appears, can be customized for individual cells and labels. The border width is specified for the entire table.

In addition, the table frame, which encloses the cells and labels, and the preview scrolling window can have the visual look of their borders customized.

Cell and Label Border Types

`XmNxrtTblBorderTypeContext` specifies the type of border drawn around cells or labels. Figure 37 shows how each border type looks.

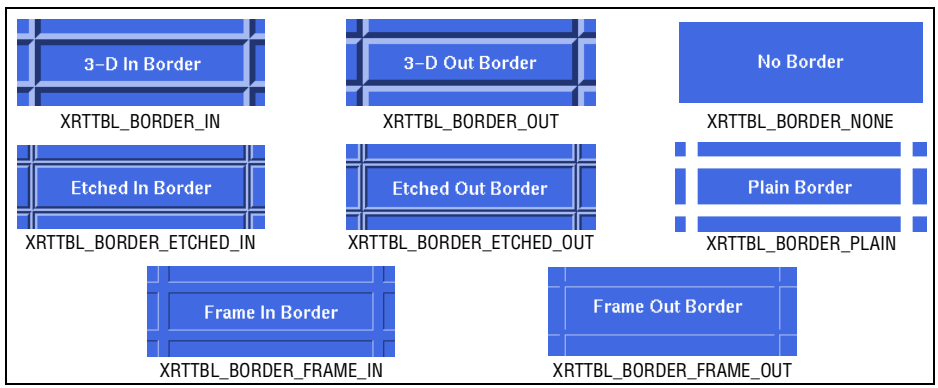


Figure 37 Border Types

The following example sets a blank border for a row:

```
XtVaSetValues(table,
    XmNxrtTblContext,           XrtTblSetContext(10, XRTTBL_LABEL),
    XmNxrtTblBorderTypeContext, XRTTBL_BORDER_NONE,
    NULL);
```

Cell and Label Border Thickness

The width of the borders around the cells and labels is specified by `XmNxrtTblShadowThickness`. This resource applies to the entire table. By default, the

borders are 2 pixels wide. Figure 38 shows the visual effect of different border widths.

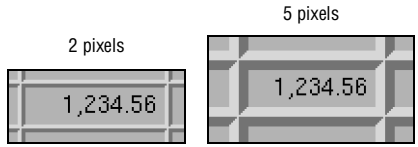


Figure 38 Setting Border Thickness

Cell and Label Border Sides

`XmNxrtTblBorderSidesContext` specifies the sides of a cell or label that display the border type (specified by `XmNxrtTblBorderTypeContext`). By default, the border type is displayed on all sides of a cell/label. Figure 39 illustrates one of the visual effects that can be achieved.

15 1/8	9 1/8	ISG Tech	ISGTF	10 1/2	10	10 1/4		16
28 1/2	9 1/4	Immune Rsp	IMNR	12 5/8	12	12 1/8	-1/4	823
27 1/4	13 3/8	Informix	IFMX	22	21	22	+1/2	6311

Figure 39 Customized Cell Borders

The valid values for `XmNxrtTblBorderSidesContext` are:

- `XRTTBL_BORDERSIDE_LEFT`
- `XRTTBL_BORDERSIDE_RIGHT`
- `XRTTBL_BORDERSIDE_TOP`
- `XRTTBL_BORDERSIDE_BOTTOM`
- `XRTTBL_BORDERSIDE_ALL`
- `XRTTBL_BORDERSIDE_NONE`

Specifying border sides is accomplished by OR-ing together all of the sides to have borders, for example:

```
XtVaSetValues(table,
    XmNxrtTblContext,       XrtTblSetContext(10, 10),
    XmNxrtTblBorderSidesContext,
        XRTTBL_BORDERSIDE_LEFT
        | XRTTBL_BORDERSIDE_TOP
        | XRTTBL_BORDERSIDE_BOTTOM,
    NULL);
```

Frame Border Attributes

The `XmNxrtTblFrameBorderType` resource allows you to specify the border type for the frame enclosing the cell and label areas. Its possible values are the same as those for `XmNxrtTblBorderTypeContext`, except that `XRTTBL_BORDER_FRAME_IN` and `XRTTBL_BORDER_FRAME_OUT` are not allowed for frame borders. Its default value is `XRTTBL_BORDER_NONE`.

The XmNxrtTblFrameShadowThickness resource specifies the thickness of the frame shadows surrounding the cell and label areas. Its default value is zero (no frame shadows).

XmNxrtTblFrameBorderType = **XRTTBL_BORDER_IN**
XmNxrtTblFrameShadowThickness = **2**



Employee Listing

Name	Address	Phone Number	Position	Salary
James Q. Doohan	1701 Planetia Blvd. Anytown, U.S.A.		President	$245,000
John F. Kricfalusi	983 Nickelodeon Street Anytown, U.S.A.	(999) 555-9876	Animator	$10,000
Marc Lenard	6 Gene Crescent Anytown, U.S.A.	(999) 555-1212	Telemarketer	$10 / hr.

Figure 40 A table with frame border attributes set

The shadow colors are calculated from the table's background. If you want to change the shadow colors, use a resource file; alternatively, use *XtNameToWidget()* to get the widget ID of the appropriate frame, and set the frame's XmNtopShadowColor and XmNbottomShadowColor resources.

Preview Scrolling Window Border

The default for this window border is XRTTBL_BORDER_NONE. This value is changed using the *XrtTblSetPreviewProperties()* function. See *XrtTblSetPreviewProperties()* on page 211 for more information.

3.13 Background Pixmap

XRT/table can display a pixmap in each cell or label in the table. The pixmap appears inside the margin of the cell. If the cell contains a cell value, the pixmap appears behind it in the background.[1]

Image Format

XRT/table supports pixmaps in XBM or XPM format. Images in other formats (such as TIFF, GIF, or Sun Rasterfile) can be converted to XBM or XPM for display in a table. You can experiment with the standard X bitmaps, or use the sample XPM pixmaps included in *$XRTHOME/src/table/samples*.

Specifying a Pixmap

XmNxrtTblPixmapContext specifies the pixmap to display in the background of each cell or label in the current context. This resource takes a pixmap as its value. While you can store XBM/XPM images directly in your C code, it is far easier to maintain images in files and read them from the file when needed. Pixmaps should be freed after use; XRT/table does not copy pixmaps set in cells/labels.

1. If a cell contains a widget, XRT/table sets the widget's XmNbackgroundPixmap resource to the pixmap.

XmNxrtTblClipPixmapContext can be used to specify a clipping mask for displaying the pixmap.

The easiest way to specify an image stored in a file is to use the Motif resource converter to convert the filename string to the Pixmap data type that XmNxrtTblPixmapContext requires, as shown below:

```
XtVaSetValues(table,
    XmNxrtTblContext,       XrtTblSetContext(1,1),
      XtVaTypedArg,    XmNxrtTblPixmapContext,
      XmRString, "my_image.xpm",
      strlen("my_image.xpm") + 1,
    NULL);
```

Removing a Pixmap

To remove a pixmap from the table, set XmNxrtTblPixmapContext to XmUNSPECIFIED_PIXMAP.

Pixmap Layout

The position of the pixmap within the cell is specified by the XmNxrtTblPixmapLayoutContext resource. You can center pixmaps, position them along any side of a cell, or use the pixmap as a tile pattern. Figure 41 shows how the values of this resource change the positioning of the pixmap.

When tiled, the pixmap extends to the edge of the cell/label border, ignoring the values of the XmNxrtTblMarginHeight and XmNxrtTblMarginWidth resources.

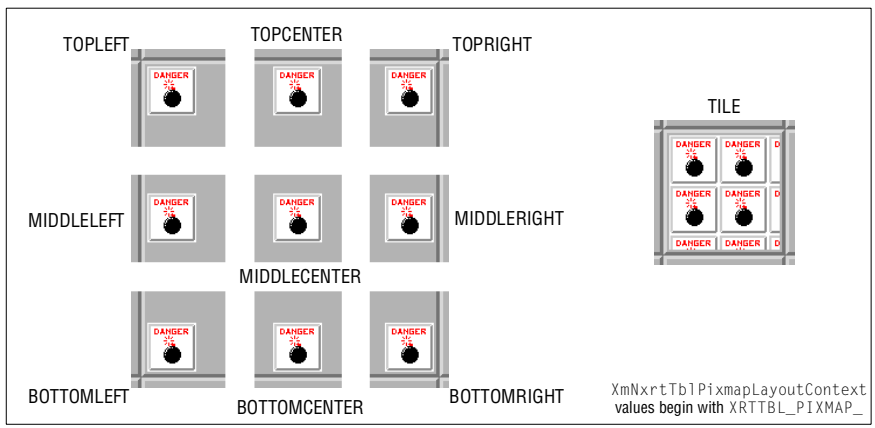


Figure 41 Pixmap layouts

Shadows and Pixmaps

When a pixmap is displayed in a cell or label, you can specify that the cell/label shadow is not to be drawn (the pixmap extends to the boundaries of the cell). To do this, set the XmNxrtTblPixmapShadowContext resource to False:

```
XtVaSetValues(table,
    XmNxrtTblContext,   XrtTblSetContext(1,1),
    XmNxrtTblPixmapShadowContext, False);
```

By default, XmNxrtTblPixmapShadowContext is True, and the cell or label shadow is drawn. (The XmNxrtTblShadowThickness resource controls the thickness of cell and label shadows.)

Specifying a Clip Pixmap

XmNxrtTblClipPixmapContext specifies the clip pixmap (clip mask) to be used when drawing a pixmap in the current context. (The clip pixmap allows you to clip, or mask, a pixmap to display only some part of it.) By default, XmNxrtTblClipPixmap is XmUNSPECIFIED_PIXMAP (no clip pixmap specified).

3.14 Text Clipping and Overflow

When cell and label values don't fit in their cells, XRT/table can either clip the display of the cell value, or allow it to flow over blank horizontally-adjacent cells. The XmNxrtTblOverflowContext resource determines which method is used. Figure 42 shows the effect of each value.

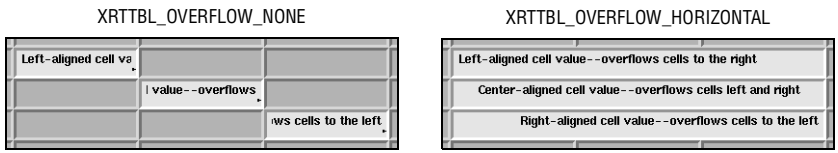


Figure 42 Cell value clipping and overflowing

Clipping

When XmNxrtTblOverflowContext is XRTTBL_OVERFLOW_NONE, a cell/label value is clipped if it does not fit in its cell in either direction.

Clipped cell values display clipping arrows at the bottom-right corner of the cell by default.[1] This provides a visual indication that there is more text in the cell than currently displayed. To turn off display of clipping arrows, set XmNxrtTblDisplayClipArrows to False.

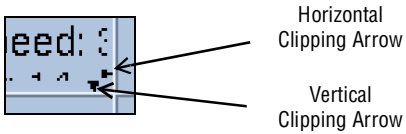


Figure 43 Display of Clipping Arrows

Overflowing

When XmNxrtTblOverflowContext is XRTTBL_OVERFLOW_HORIZONTAL, a cell/label value that does not horizontally fit in its cell flows over adjacent blank cells.

1. The current cell never displays clipping arrows because it is an XmText widget.

Overflowing is dynamic, and entirely dependent on the value of the cell/label. Any change in the table that affects the horizontal fit of the cell/label value in its cell/label can change the overflowing.

Beginning with the original (source) cell, XRT/table visually extends the cell horizontally over adjacent cells until it can horizontally display the entire cell/label value, or until it reaches a non-blank cell. The overflow direction is determined by `XmNxrtTblAlignmentContext`, shown in Figure 44.

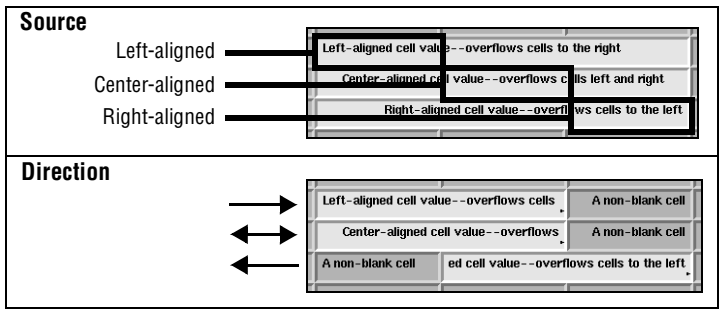


Figure 44 Overflow source and direction

The overflowed cell appears to be one joined cell; this is an illusion. The adjacent cells are still traversable and editable. If a user enters a value for a cell underneath part of an overflowed cell, the overflow is clipped at that cell. To edit an overflowed cell, the user must traverse to the cell that is the source of the overflow (see Figure 44).

Care should be taken when using cell overflowing and CellValue/LabelCreate callbacks (specified by `XmNxrtTblCellValueCallback` and `XmNxrtTblLabelCreateCallback`) in the same table, because cells/labels that overflow may obscure the cell/label values set by your callbacks.

Word Wrapping

XRT/table allows you to specify that text in cells should be displayed as word wrapped. In this case, line breaks in a cell occur only on a space, tab or newline. The cell/label value will still clip or overflow if it cannot be completely displayed.

To specify word wrapping, set `XmNxrtTblWordWrapContext` to `True`. By default, `XmNxrtTblWordWrapContext` is `False`, and word wrapping is not performed.

Overflow vs. Spanning

Cell overflow appears visually similar to cell spanning (discussed in the next section) but they have different purposes and assumptions. Overflow is useful in spreadsheet-type applications, where the user has a lot of control over the cell values themselves. It's also useful when you'd like column labels to overflow horizontally when necessary. If you need to overflow pixmaps or widgets, or you are designing complex forms, you should use spanning.

3.15 Cell and Label Spanning

Spanning is a way to join a range of cells or labels together and treat them as one cell/label. A spanned range looks and acts like one cell/label that covers several rows and/or columns. There are many potential uses for spanning, including designing complex forms and displaying large pixmaps or widgets.

When you create a spanned range, the top-left cell in the range is extended over the entire range. The top-left cell is the source cell, and its value and attributes apply over the entire span, covering any values or attributes set for the other cells/labels in the range.

Spanned ranges must begin at the top-left corner of the range. A span cannot contain both cells and labels.

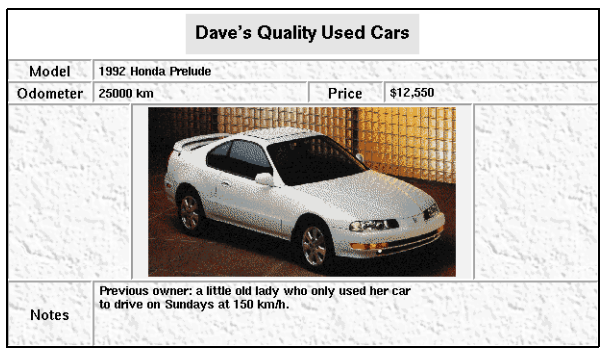


Figure 45 Form design using spanned cells

The `XmNxrtTblSpanList` resource specifies the list of all currently spanned ranges; the last element of this list must be `NULL`. Each range in the list is an `XrtTblRange` structure:

```
typedef struct {
    int     start_row, start_column;
    int     end_row, end_column;
} XrtTblRange;
```

The following example defines two spanned ranges:

```
static XrtTblRange span1 = {0, 0, 4, 2};
static XrtTblRange span2 = {7, 1, 7, 4};
static XrtTblRange *all_spans[] =
    {&span1, &span2, NULL};

XtVaSetValues(table,
        XmNxrtTblSpanList,  all_spans,
        NULL);
```

Adding a Span

To add a range to the current spanned range list, use *XrtTblRangeListAdd()*. This procedure has the following prototype:

```
XrtTblRange **
XrtTblRangeListAdd(
    XrtTblRange     **list,
    int             position,
    XrtTblRange     *range
    )
```

XrtTblRangeListAdd() returns a new range list that includes the range you added to it. You can set `XmNxrtTblSpanList` to this list. The following example adds a current selection to the spanned range list:

```
XrtTblRange selection, **span_list;

if (!XrtTblGetSelectedRange(table, 0, &selection))
    return;

XtVaGetValues(table, XmNxrtTblSpanList, &span_list, NULL);
span_list = XrtTblRangeListCopy(span_list);
span_list = XrtTblRangeListAdd(span_list, XRTTBL_MAXINT,
         &selection);

XtVaSetValues(table,
    XmNxrtTblSpanList,          span_list,
    XmNxrtTblSelectedCellList,  NULL,
    NULL);
XrtTblRangeListFree(span_list);
```

Working with Spanned Ranges

All Context resources and other attributes must be set on the source (top-left) cell of a spanned range. You can set and retrieve attributes for the other cells in the span, but since the cells are spanned with the source cell, you won't see any change.

You can determine whether a cell or label is currently part of a spanned range by calling *XrtTblGetSpanByRowCol()*.

Removing a Span

To remove a spanned range from the current list, use *XrtTblRangeListDelete()* on a copy of the spanned range list and set `XmNxrtTblSpanList` to the returned range list.

Set `XmNxrtTblSpanList` to `NULL` to remove all spanned ranges.

3.16 Cell and Label Flashing

When a table contains important data that you want to highlight to end-users, it is often effective to flash the colors of cells. XRT/table provides the ability to flash different table contexts with the *XrtTblFlash()* method. The current cell does not flash, even if the cell traversed to is a flashing cell (the flashing border is visible though).

The prototype of *XrtTblFlash()* is as follows:

```
Boolean
XrtTblFlash(
    Widget          table,
    XrtTblContext   *context,
    int             count,
    String          bg_color,
    String          fg_color,
    unsigned long   on_duration,
    unsigned long   off_duration
    )
```

The method takes a context, the number of times to flash, the colors to use for the flashing, and the duration of each portion of the flash. Setting a flash overrides any previous flashing for that context.

The approximate duration of flashing is determined by the following formula:

$$\text{total duration} = \textit{count} * (\textit{on_duration} + \textit{off_duration})$$

The *on_duration* and *off_duration* parameters each specify the number of milliseconds to hold its portion of the flash for. The *count* parameter specifies the number of times to flash (on and off). If *count* is `XRTTBL_MAXINT`, flashing continues infinitely.

The following effects can be achieved by combining different foreground and background flash colors:

- Flash opposite colors: set *bg_color* to `NULL` to use the cell's foreground color; pass *fg_color* as `NULL` to use the cell's background color.
- Flash text only: set *bg_color* to the same background color as the context and *fg_color* to the color you want the text to flash.
- Flash background only: set *fg_color* to the same foreground color as the context and *bg_color* to the color you want the background to flash.

The following example flashes all row labels an infinite number of times:

```
XrtTblFlash(table,
        XrtTblSetContext(XRTTBL_LABEL, XRTTBL_ALL),
        XRTTBL_MAXINT, "red", "yellow", 100, 100);
```

Note: If multiple cells are set to flash with the same on/off duration, XRT/table synchronizes the flashing.

Stopping Flashing

To stop flashing for one context, call *XrtTblFlash()* with the *count* argument set to 0. All other parameters are ignored.

To stop all flashing in the entire table, call *XrtTblFlash()* with *context* set to the entire table and *count* set to 0, for example:

```
XrtTblFlash(table,
        XrtTblSetContext(XRTTBL_ALL, XRTTBL_ALL),
        0, NULL, NULL, 0, 0);
```

3.17 User Data

The XmNxrtTblUserDataContext resource allows an application to store any data for any context of cells or labels. XRT/table does not make any internal use of this resource. This resource is useful for attaching secondary data to a part of the table, such as the time the cell was last edited.

3.18 Table Printing

Many applications need to provide end-users with a way to get a printout of a table they see on screen. XRT/table can generate a PostScript image of a table. This image is created with device-independent PostScript fonts and operators. The PostScript file can be printed on any PostScript printer or imported into other documents.

3.18.1 Generating PostScript Output

XRT/table provides two methods that generate PostScript table images: *XrtTblDrawPS()* and *XrtTblVaDrawPS()*. The two methods produce the same output, but provide different usage interfaces. *XrtTblVaDrawPS()* uses a *VarArgs*-style interface (similar to *XtVaSetValues()*) to allow you to set only the arguments that you need to change from their defaults. *XrtTblDrawPS()* uses an *ArgList*-style interface to allow you to maintain different sets of arguments for different printing requirements. Both methods are described in Appendix B on page 191, but the arguments are described in detail only for *XrtTblVaDrawPS()*.

If the table has a title or other label widget, it will only be printed if it is managed by the table widget.

Defining Headers and Footers

To define headers and footers for the pages on which your table is printed, set the XRTTBL_PS_HEADER and XRTTBL_PS_FOOTER attributes when calling *XrtTblDrawPS()* or *XrtTblVaDrawPS()*.

The fonts for these headers and footers are controlled by the XRTTBL_PS_HEADER_FONT and XRTTBL_PS_FOOTER_FONT attributes; the header and footer margins are controlled by XRTTBL_PS_HEADER_MARGIN and XRTTBL_PS_FOOTER_MARGIN.

Usage Tips

Following are some tips on using *XrtTblDrawPS()* and *XrtTblVaDrawPS()*:

- The XRTTBL_PS_PAPERSIZE_ and XRTTBL_PS_MARGIN_ attributes are always relative to the top of the *paper*, not the top of the *image*. That is, if the table image orientation is landscape, the XRTTBL_PS_PAPERSIZE_TOP still measures the top of the page, not the top of the image.
- To print multiple table widgets on one page, explicitly position each table on the page (so they don't overlap) with the XRTTBL_PS_MARGIN_ attributes. Set XRTTBL_PS_SHOWPAGE to False for every image except the last one.

- If you specify XRTTBL_PS_BEST for XRTTBL_PS_ORIENTATION, be sure the way it works is explained to end-users. Since it analyzes the table image to be printed each time it runs, differently-sized ranges of the same table will not always print at the same image orientation.

Limitations

The following discusses the known limitations of the printing methods:

- The methods print at least one row/column on each page. If, due to the size of cells or scaling set when printing, a row/column does not fit on one page, it is clipped on the PostScript output.
- Use the automatic color determination feature (XRTTBL_PS_COLOR set to XRTTBL_PS_AUTO) with care. Some monochrome PostScript printers (such as the SPARCprinter) incorrectly identify themselves as color printers. To force monochrome output on these printers, un-comment the "/color? false def" line in the PostScript file.
- XRT/table is optimized for printing small, repeating pixmaps, on the order of 64x64 pixels. Performance may suffer if large pixmaps are printed.

XrtTblVaDrawPS() example

The following code outputs a range to a file, with page breaks specified on several columns, in landscape orientation. All other attributes default.

```
FILE *fp;
int col_pagebreaks[] = { 5, 10, 25, -1 };
int numpages;
XrtTblRange range = { 10, 0, 40, 30 };
fp = fopen("mytable.ps", "w");
if (fp) {
    numpages = XrtTblVaDrawPS(table, fp,
                XRTTBL_PS_ORIENTATION,       XRTTBL_PS_LANDSCAPE,
                XRTTBL_PS_COL_PAGEBREAKS,    col_pagebreaks,
                XRTTBL_PS_SHOW_FROZEN_COLS,  XRTTBL_PS_ALL,
                XRTTBL_PS_SHOW_FROZEN_ROWS,  XRTTBL_PS_ALL,
                XRTTBL_PS_CELL_RANGE,        &range,
                NULL);
    printf("%d page%s written.\n", numpages,
                (numpages==1)? "" : "s");
}
else {
    printf("error opening file\n");
}
```

XrtTblDrawPS() example

The following example builds the argument list needed to print a table on European A4 paper and calls *XrtTblDrawPS()*:

```
int ac, num_pages;
FILE *fp;
double scale = 100.0;
double a4_width  = 21.0;
double a4_height = 29.7;
double a4_margin = 1.0;
XrtTblPSarg al[20];

num_pages = 0;
fp = fopen("mytable.ps", "w");
if (fp) {
    ac = 0;
    XtSetArg(al[ac], XRTTBL_PS_SCALE, &scale); ac++;
    if (paper_type == METRIC_A4){
        XtSetArg(al[ac], XRTTBL_PS_UNITS, XRTTBL_PS_CM); ac++;
        XtSetArg(al[ac], XRTTBL_PS_PAPERSIZE_HEIGHT, &a4_height);
            ac++;
        XtSetArg(al[ac], XRTTBL_PS_PAPERSIZE_WIDTH, &a4_width);
            ac++;
        XtSetArg(al[ac], XRTTBL_PS_MARGIN_TOP, &a4_margin); ac++;
        XtSetArg(al[ac], XRTTBL_PS_MARGIN_BOTTOM, &a4_margin); ac++;
        XtSetArg(al[ac], XRTTBL_PS_MARGIN_LEFT, &a4_margin); ac++;
        XtSetArg(al[ac], XRTTBL_PS_MARGIN_RIGHT, &a4_margin); ac++;
    }
    num_pages = XrtTblDrawPS(table, fp, al, ac);
    fclose(fp);
}
printf("%d page%s written.\n", num_pages,
                    (num_pages == 1) ? "" : "s");
}
```

Sample Print Pop-up

The source code for a sample print pop-up window that shows one way to use *XrtTblVaDrawPS()* and how to pipe the file to *lp* is located in *$XRTHOME/src/table/tools/pdbox_table.c*.

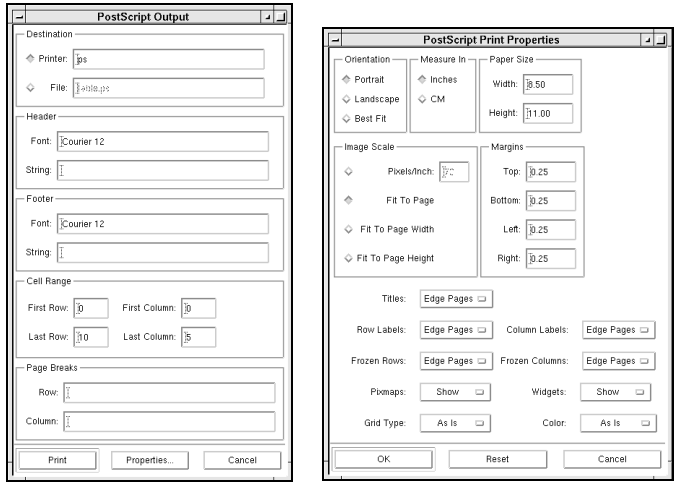


Figure 46 Print Dialog Boxes from sample Print Pop-up Code

3.18.2 Customizing PostScript Fonts

Most X Servers provide enough information about the X fonts used in the table to allow the PostScript output methods to automatically choose appropriate PostScript fonts. However, you can define a list of global PostScript to X font correspondences using the `XmNxrtTblPSFontMapList` resource. Each element of the list must be of type `XrtTblPSFontMap`, which is defined as follows:

```
typedef struct {
    String  font_name;
    String  tag;
    String  PS_font;
    double  point_size;
} XrtTblPSFontMap;
```

In this structure, *PS_font* is the PostScript font, and *point_size* is its point size. (Figure 47 lists the of standard PostScript font names.) The X font can be specified in either of two ways:

- You can use *font_name* to specify a complete X font name, such as "`-*-times-bold-r-normal-*-140-*`".
- You can use *tag* to specify an `XmNfontList` font tag (such as `bold`).

The list is terminated by defining an element with *PS_font* set to `NULL`.

If *point_size* is set to 0, the point size of the PostScript font is inferred from the point size of the corresponding X font.

```
Times-Roman           Helvetica                NewCenturySchlbk-Roman
Times-Bold            Helvetica-Bold           NewCenturySchlbk-Bold
Times-Italic          Helvetica-Oblique        NewCenturySchlbk-Italic
Times-BoldItalic      Helvetica-BoldOblique    NewCenturySchlbk-BoldItalic

Palatino-Roman        Bookman-Light            Helvetica-Narrow
Palatino-Bold         Bookman-Demi             Helvetica-Narrow-Bold
Palatino-Italic       Bookman-LightItalic      Helvetica-Narrow-Oblique
Palatino-BoldItalic   Bookman-DemiItalic       Helvetica-Narrow-BoldOblique

Courier               AvantGarde-Book          Symbol
Courier-Bold          AvantGarde-Demi          ZapfChancery-MediumItalic
Courier-Oblique       AvantGarde-BookOblique   ZapfDingbats
Courier-BoldOblique   AvantGarde-DemiOblique
```

Figure 47 The 35 Standard LaserWriter II Font Names

The following example uses `XmNxrtTblPSFontMapList` to specify that Helvetica Narrow PostScript fonts are to correspond to Helvetica X fonts:

```
XrtTblPSFontMap fontlist[] = {
    {"-*-helvetica-medium-r-*-*-*-*-*-*-*-*-iso8859-1",
        NULL, "Helvetica-Narrow", 0},
    {"-*-helvetica-bold-r-*-*-*-*-*-*-*-*-iso8859-1",
        NULL, "Helvetica-Narrow-Bold", 0},
    {"-*-helvetica-medium-o-*-*-*-*-*-*-*-*-iso8859-1",
        NULL, "Helvetica-Narrow-Oblique", 0},
    {"-*-helvetica-bold-o-*-*-*-*-*-*-*-*-iso8859-1",
        NULL, "Helvetica-Narrow-BoldOblique", 0},
    {NULL, NULL, NULL, 0}
};
...
XtVaSetValues(table,
    XmNxrtTblPSFontMapList,fontlist,
    NULL);
```

3.18.3 Print Callbacks

The `XmNxrtTblPrintCallback` resource specifies a list of callbacks that are called before and after each page of a table is printed. These callbacks allow you to monitor the printing process and change the output print file if desired. Each callback in this list is passed the following structure.

```
typedef struct }
    int          reason;             // Read-only
    XEvent       *event;             // Read-only
    XrtTblRange  **range;            // Read-only
    int          current_page;       // Read-only
    int          total_pages;        // Read-only
    int          across_pages;       // Read-only
    int          down_pages;         // Read-only
    int          page_number;
    String       header, footer;
    FILE         *fp;
    Boolean      write_file_header;
    Boolean      write_file_trailer;
    Boolean      doit;               // initially True
} XrtTblPrintCallbackStruct;
```

reason is either `XRTTBL_REASON_PRINT_BEGIN` (before page printed) or `XRTTBL_REASON_PRINT_END` (after page printed). *range* is the range of cells on the page, *current_page* is the page being printed, and *total_pages* is the total number of pages to print (measured as *across_pages* multiplied by *down_pages*).

The following members of this structure can be modified to allow you to control the table printing process:

- *header* and *footer* are the page header and footer (specified by the call to *XrtTblVaDrawPS()*).
- *page_number* contains the page number displayed in the header or footer.
- *fp* is the file to which the table is being written. (If you change this value to a new file, you will need to set *write_file_header* to `True` when you write to this new file for the first time. This tells XRT/table to reprint the PostScript file header.) Changing *fp* enables you to, for example, write even and odd-numbered pages to separate files.
- If you are writing the last page of a file and the page is not the last page of the table, you will need to set *write_file_trailer* to True. This tells XRT/table to write the PostScript file trailer. *write_file_trailer* is automatically set to `True` if *current_page* equals *total_page* (if the page being printed is the last page of the table).
- *doit* controls whether the current page is to be printed. To skip a page, set *doit* to `False` when *reason* is `XRTTBL_REASON_PRINT_BEGIN`. (Setting *doit* has no effect on *current_page* or *total_pages*.)

Print Widget Callbacks

Your application can use the `XmNxrtTblPrintWidgetCallback` resource to define a list of callbacks that are called before and after a widget in a cell/label is printed. This enables you to skip printing large or complicated widgets. Each callback in this list is passed the following structure:

```
typedef struct {
    int         reason;         // Read-only
    XEvent      *event;         // Read-only
    Widget      widget;         // Read-only
    int         row, column;    // Read-only
    FILE        *fp;            // Read-only
    int         current_page;   // Read-only
    int         page_number;    // Read-only
    Boolean     doit;           // initially True
} XrtTblPrintWidgetCallbackStruct;
```

The *reason* member is either `XRTTBL_REASON_PRINTWIDGET_BEGIN` (before printing) or `XRTTBL_REASON_PRINTWIDGET_END` (after printing). *widget* specifies the widget to be printed, and *row* and *column* indicate its location in the table. If you do not want to print the widget, set *doit* to `False`.

All of the other members of this structure behave as they do in Print callbacks (described in the previous section). None of the members can be modified from within PrintWidget callbacks.

4

XRT/table Data

The XrtTblCellValues Structure ■ *Getting Data into a Table*
Manipulating Table Data ■ *Using Multiple XrtTblCellValues Structures*
Cell Value Callbacks ■ *Getting Data out of the Table*

This chapter describes the `XrtTblCellValues` structure in detail and describes how to allocate and manipulate the structure.

4.1 The XrtTblCellValues Structure

The `XrtTblCellValues` structure is a complex, dynamically-managed data structure designed to efficiently store large amounts of tabular text data.

When XRT/table draws or redraws each cell,[1] it retrieves the cell value from the `XrtTblCellValues` structure pointed to by `XmNxrtTblCellValues` (this pointer and the structure it points to should be considered *read-only*). If there is no structure, or no value exists in the structure for the cell, XRT/table draws a blank cell.[2]

As a program or a user enters cell values, XRT/table automatically updates and maintains its `XrtTblCellValues` structure.

This structure is not tied to the table's attributes. `XmNxrtTblCellValues` can be set to display several `XrtTblCellValues` structures without affecting the attributes displayed (such as cell background colors and fonts). Figure 48 illustrates the separation of cell values from the table.

1. XRT/table draws cells as a result of scrolling or expose events.

2. If a cell value method has been defined, the table calls that method for the value to display for blank cells.

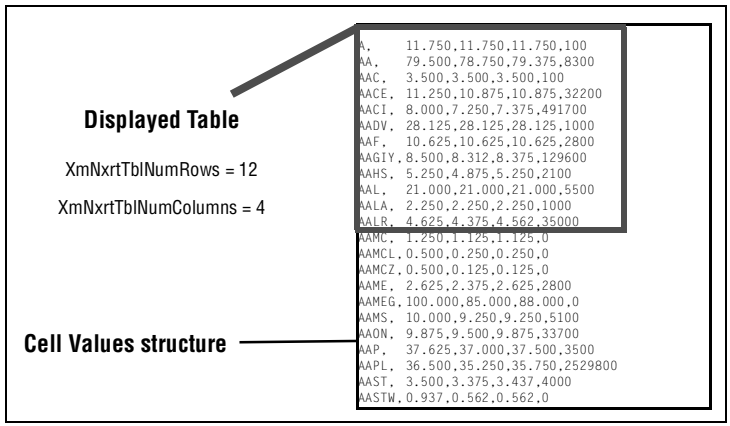


Figure 48 Relationship of XrtTblCellValues Structure to Table Attributes

Structure Definition

The `XrtTblCellValues` structure is defined as follows:

```
typedef struct {
    int rows;
    XrtTblXmStringSet set_mode;
    XrtTblXmStringList **values;
} XrtTblCellValues;
```

values is the list of values. *set_mode* indicates whether *values* contains XmStrings or Strings; it must be set to either `XRTTBL_SET_STRING` (set using Strings) or `XRTTBL_SET_XMSTRING` (set using XmStrings). The procedures and methods described above that create `XrtTblCellValues` structures set *set_mode* for you. By default, *set_mode* is set to `XRTTBL_SET_STRING`.

Convenience Macros

There are a number of macros in *<Xm/XrtTable.h>* that facilitate accessing portions of the `XrtTblCellValues` structure (and other parts of the table).

`CELL_VALUE` retrieves a cell value from an `XrtTblCellValues` structure. Other macros allow you to check whether a cell contains a value, or if a context is a cell or label. See Appendix C on page 221 for a full listing of the macros.

Convenience Procedures

There are a number of procedures and methods that manipulate `XrtTblCellValues` structures. These are fully described in Appendix B on page 191. The source code for some procedures is in *$XRTHOME/src/table/util*.

XrtTblAddColumns(), XrtTblAddRows()	Inserts new rows/columns into the table's `XrtTblCellValues` structure and increases `XmNxrtTblNumRows` and `XmNxrtTblNumColumns`. Keeps the attributes of existing rows/columns together with their cell values.
XrtTblDeleteColumns(), XrtTblDeleteRows()	Removes rows/columns from the table's `XrtTblCellValues` structure and their attributes. Decreases `XmNxrtTblNumRows` and `XmNxrtTblNumColumns`.
XrtTblMoveColumns(), XrtTblMoveRows()	Moves rows or columns within the table's `XrtTblCellValues` structure. Keeps the attributes of existing rows/columns together with their cell values.
XrtTblGetCell()	Gets the value from the table, `or resulting from a cell value function or CellValue callback`. A string is returned if `XmNxrtTblXmStringSetMode` is `XRTTBL_SET_STRING`.
XrtTblGetCellValues()	Creates a new `XrtTblCellValues` structure and loads it with values copied from the table's `XrtTblCellValues` structure.
XrtTblSetCell()	Sets the cell value, optionally validating and redrawing the cell. Motif 1.2 applications can use a String or XmString. Strings can also be specified if `XmNxrtTblXmStringSetMode` is set appropriately. This method allows the update of a single cell as fast as possible; hence, it does not perform any cell resizing.
XrtTblSetCellValues()	Copies cell values from a supplied `XrtTblCellValues` structure to the table's `XrtTblCellValues` structure.
XrtTblCellValuesCopy()	Creates a copy of a cell values structure.
XrtTblSortTable()	Sorts rows in the table's `XrtTblCellValues` structure based on a column. Callbacks can be specified after each call to *XrtTblSortTable()* using the `XmNxrtTblSortCallback` resource.

4.2 Getting Data into a Table

Perhaps the most difficult task facing an XRT/table developer is getting the data to be displayed and inserting it into an `XrtTblCellValues` structure for display in a table.

Data from a File (static)

If the data can be formatted in a file, use *XrtTblReadAscii()* to create an `XrtTblCellValues` structure and read the values into it. This method reads data formatted as rows of comma-separated values, and is fully described in Appendix B on page 191. Figure 3 on page 15 shows an example of its use. The source code for this method is provided in *$XRTHOME/src/table/util*. You can modify a copy of this method to support your particular data format.

Note: You should set `XmNxrtTblNumRows` and `XmNxrtTblNumColumns` to the size of the cell values structure when using *XrtTblReadAscii()* because the cell values structure is independent of the number of rows/columns in the table. The following code shows how to do this:

```
int last_row, last_column;
...
values = XrtTblReadAscii(toplevel, fp, ',');
XrtTblGetCellValuesSize(values, &last_row,
        &last_column, NULL);
table = XtVaCreateManagedWidget("table",
    xtXrtTableWidgetClass,      toplevel,
    XmNxrtTblNumRows,           last_row + 1,
    XmNxrtTblNumColumns,        last_column + 1,
    ...
    NULL);
```

Another approach is to populate the `XrtTblCellValues` structure after the widget has been created using `XmNxrtTblCellValueContext`. XRT/table automatically allocates and creates an `XrtTblCellValues` structure if there is not one attached to it when the widget is created.

Data that Changes (dynamic)

Data that changes dynamically needs more careful coding. The Xt Intrinsics provides procedures that can be used to trigger callbacks to deal with runtime events. In particular, *XtAppAddInput()* can be used to register an application callback routine that is called whenever there is input pending on a file descriptor. The file descriptor can represent a file, pipe, or even a socket connection to a process running on another machine. Your callback routine can use `XmNxrtTblCellValueContext` to add this data to the table's `XrtTblCellValues` structure.

Another approach is for the application to use *XtAddTimeOut()* to define an interval timer. Whenever the specified time interval expires, the application's callback routine will be executed. At this time, your callback routine can check to see if there is any data to add to the table and add it using `XmNxrtTblCellValueContext`.

4.3 Manipulating Table Data

This section discusses manipulating data within the XrtTblCellValues structure pointed to by XmNxrtTblCellValues.

4.3.1 Setting and Retrieving Cell Values

Setting Cell Values

To specify cell values in the current XrtTblCellValues structure, set the context and the XmNxrtTblCellValueContext resource as shown by the following example:

```
String values[] =
       { "value 1", "value 2", "a multi-\nline value", NULL };
for (i=0; values[i]; i++)
    XtVaSetValues(table,
            XmNxrtTblContext,           XrtTblSetContext(20, i),
            XmNxrtTblCellValueContext,  values[i],
            NULL);
```

By default, XmNxrtTblCellValueContext expects values of type XmString, but accepts values of type String. XRT/table automatically converts String values to XmString.

To update a single cell as fast as possible, use the *XrtTblSetCell()* method. This sets the value of the indicated cell and optionally validates and redraws the cell. As with XmNxrtTblCellValueContext, *XrtTblSetCell()* expects values of type XmString, but accepts values of type String. Note that this routine does not resize the cell to fit the new data, even if XmNxrtTblXmStringSetMode is set appropriately.

To specify the value type used when setting cells (and labels), use the XmNxrtTblXmStringSetMode resource.[1] This resource can be set to the following:

XRTTBL_SET_EXAMINE	value is an XmString or String (default)[2]
XRTTBL_SET_XMSTRING	value is an XmString
XRTTBL_SET_STRING	value is a String

The XmNxrtTblXmStringSetMode resource can only be set at table creation time.

User Editing

When a user edits a cell value, that value replaces the previous cell value. Unless you disable cell editing, users can change any cell values set by the application. See section 5.7 on page 96 for information on controlling interactive editing.

1. This resource also determines the data type of row and column label lists (XmNxrtTblRowLabels and XmNxrtTblColumnLabels, and the type for cell/label value callbacks (XmNxrtTblCellValueCallback and XmNxrtTblLabelCreateCallback).

2. XmNxrtTblXmStringSetMode cannot be set to XRTTBL_SET_EXAMINE if you are using Motif 2.x.

Retrieving Cell Values

There are two ways to retrieve cell values from the table's XrtTblCellValues structure:

- Use XmNxrtTblCellValueContext to get the value of a single cell (current context must be a one cell), for example:

```
XmString value1;
XtVaSetValues(table,
    XmNxrtTblContext, XrtTblSetContext(0,0),
    NULL);
XtVaGetValues(table,
    XmNxrtTblCellValueContext, &value1,
    NULL);
value1 = XmStringCopy(value1);
```

or

- Use XrtTblGetCell() to get the value of the indicated cell from a table, if present, or from XmNxrtTblCellValueFunc or XmNxrtTblCellValueCallback, for example:

```
int row=1,col=1;
XmString value1;
value1=XrtTblGetCell(table,row,col);
value1=XmStringCopy(value1);
```

or

- Use CELL_VALUE to get the value of a cell at a location, for example:

```
XrtTblCellValues *cells;
String string;
int i, j, rows, columns;
XmString value;

XtVaGetValues(table, XmNxrtTblCellValues, &cells, NULL);
XrtTblGetCellValuesSize(cells, &rows, &columns, NULL);
for (i=0; i <= rows; i++) {
    for (j=0; j <= columns; j++) {
        if ((value = CELL_VALUE(cells, i, j))) {
            string = XrtTblCvtXmStringToString(value));
            printf("%d %d: %s\n", i, j, string);
            XtFree(string);
        }
    }
}
```

By default, a cell value is retrieved as an XmString. To retrieve cell values as Strings, set the XmNxrtTblXmStringGetMode resource to XRTTBL_GET_STRING.

Specifying the Value Type

The XmNxrtTblXmStringGetMode resource allows you to specify the data type of values retrieved from cells (and labels), as well as the type of row and column label lists. This resource can be set to the following:

XRTTBL_GET_XMSTRING	get as XmString (default)

XRTTBL_GET_STRING	get as String

Note: If you are linking to Motif 2.x, you cannot set XmNxrtTblXmStringSetMode to XRTTBL_SET_EXAMINE. If an application tries to do this, an error message will be printed, and the application will abort.

4.3.2 Adding Rows and Columns

XrtTblAddColumns() and *XrtTblAddRows()* insert new rows and columns into the table, optionally filling the new cells with values.[1] *XrtTblAddColumns()* shifts existing cell values and attributes (such as, for example, background color) to the right. *XrtTblAddRows()* shifts existing cell values and attributes down. You can specify whether to shift existing labels along with the rows or columns shifted. XmNxrtTblNumRows or XmNxrtTblNumColumns is updated to reflect the insertion.

When using *XrtTblAddColumns()* and *XrtTblAddRows()*, note the following:

- If you do not supply values for the new cells within the method, they are blank. Values for the new row or column labels must be specified separately. The starting row or column cannot be greater than XmNxrtTblNumRows or XmNxrtTblNumColumns.
- When no values are specified for a new row or column having a variable height/width, the row/column does not appear–the row/column is hidden. For more information on variable rows/columns, see section 3.7 on page 48.
- If the current cell is at or below the starting row/column, any cell editing is cancelled, and the XmText widget is unmapped.
- Values can be specified as either an array of XmStrings or an array of Strings. The XmNxrtTblXmStringSetMode resource specifies which is expected.

The prototype for *XrtTblAddColumns()* is as follows:

```
Boolean
XrtTblAddColumns(
    Widget  table,
    int     position,
    int     num_columns,
    Boolean shift_labels,
    void    **values,
    int     num_values
    )
```

Appendix B contains a detailed description of the method's arguments. The following example inserts 5 columns before column 0, with the first column populated with 3 values:

```
String values[] = { "a", "b", "c" };
XrtTblAddColumns(table, 0, 5, False, values, 3);
```

1. To save a copy of a structure before changing it, use *XrtTblCellValuesCopy()*.

4.3.3 Deleting Rows and Columns

XrtTblDeleteColumns() and *XrtTblDeleteRows()* delete rows and columns from the table. *XrtTblDeleteColumns()* shifts existing cell values and attributes to the left. *XrtTblDeleteRows()* shifts existing cell values and attributes up. You can specify whether to shift existing labels along with the shifted rows or columns. `XmNxrtTblNumRows` or `XmNxrtTblNumColumns` is updated to reflect the deletion.

When using *XrtTblDeleteColumns()* and *XrtTblDeleteRows()*, note the following:

- The starting row or column cannot be greater than `XmNxrtTblNumRows` or `XmNxrtTblNumColumns`.
- If the current cell is at or below the starting row or column, any cell editing is cancelled, and the XmText widget is unmapped.
- Decreasing `XmNxrtTblNumRows` or `XmNxrtTblNumColumns` does not *delete* rows or columns from the table; it only decreases the scrollable region.

The prototype for *XrtTblDeleteColumns()* is as follows:

```
Boolean
XrtTblDeleteColumns(
    Widget  table,
    int     position,
    int     num_columns,
    Boolean shift_labels
    )
```

Appendix B contains a detailed description of the method's arguments. The following call deletes rows 2 to 5, and shifts the row labels up along with the rows:

```
XrtTblDeleteRows(table, 2, 4, True);
```

4.3.4 Moving Rows and Columns

XrtTblMoveColumns() and *XrtTblMoveRows()* moves rows and columns (and optionally the labels) within the table. *XrtTblMoveColumns()* and *XrtTblMoveRows()* shift existing cell values and attributes (such as background color) affected by the move. You can specify whether to move existing labels along with the rows or columns moved.

The source or destination row or column cannot be greater than the corresponding `XmNxrtTblNumRows` or `XmNxrtTblNumColumns` value.

If the current cell is at or below any affected row or column, any cell editing is cancelled, and the XmText widget is unmapped.

The prototype of *XrtTblMoveColumns()* is as follows:

```
Boolean
XrtTblMoveColumns(
    Widget  table,
    int     source,
    int     num_columns,
    int     destination,
    Boolean shift_labels
    )
```

Appendix B contains a detailed description of the method's arguments. The following call moves rows 4 to 6 to start before row 1, without shifting the labels:

```
XrtTblMoveRows(table, 4, 3, 1, False);
```

4.3.5 Clearing Cell Values

You can easily clear the cell values for cells, rows, columns, or the entire table, by setting `XmNxrtTblCellValueContext` to `NULL` for that context. The visual attributes of the cells do not change.

4.3.6 Clearing the Entire Table

To clear the table of all cell values, set `XmNxrtTblCellValues` to `NULL`. The `XmNxrtTblNumRows` and `XmNxrtTblNumColumns` resources do not change, and any attributes defined for the table remain in effect.

4.4 Using Multiple XrtTblCellValues Structures

XRT/table makes it easy to change all cell values while leaving the table's visual attributes unchanged. Applications can create several `XrtTblCellValues` structures and then switch between them by setting `XmNxrtTblCellValues` to point to one of the cell values structures.

To replace the table's cell values with a different `XrtTblCellValues` structure, first create the new structure in one of the following ways:

- Call *XrtTblReadAscii()* to create and load a new `XrtTblCellValues` structure with values from a file.
- Call *XrtTblGetCellValues()* to create and load a new `XrtTblCellValues` structure with values from the table.
- Call *XrtTblGetGraphData()* or *XrtTblGet3dData()* to create and load a new `XrtTblCellValues` structure with data converted from an XRT/graph or XRT/3d dataset.

Then set the `XmNxrtTblCellValues` resource to point to the newly-created structure. After XRT/table copies the structure to its internal memory, you may destroy your copy. The following example illustrates this sequence:

```
XrtTblCellValues *values;
...
values = XrtTblReadAscii(toplevel, fp, ',');
XtVaSetValues(table,
    XmNxrtTblCellValues,         values,
    NULL);
XrtTblDestroyCellValues(values);
```

4.5 Cell Value Callbacks

Another way to set cell values is using a CellValue callback. This requires more work in the application but improves the performance of large tables.

The `XmNxrtTblCellValueCallback` resource specifies a callback routine that XRT/table calls at runtime when it first displays any empty cell. The application is completely responsible for performing any addition, deletion, moving, or sorting of the cell values that may be required.

The callback you create can be set to allocate the returned XmString (XRT/table frees it) or pass a static pointer, and is passed the following structure:

```
typedef struct {
    int             reason;      // Read-only
    XEvent          *event;      // Read-only
    int             row, column; // Read-only
    XmString        string;
    Boolean         store;       // initially False
    Boolean         free_string; // initially True
} XrtTblCellValueCallbackStruct;
```

The following is an example of a simple CellValue callback:

```
static void
cellvalueCB(Widget table, XtPointer data,
    XrtTblCellValueCallbackStruct *cbs)
{
    static char buf[30];
    sprintf(buf, "Row: %d, Column: %d",
            cbs->row, cbs->column);
    cbs->string = (XmString) buf;
    cbs->free_string = False;
}
```

This example uses a static buffer to store the cell value, and sets the *free_string* member of the `XrtTblCellValueCallbackStruct` to tell XRT/table not to free the cell value when finished with it.

To add this callback to the CellValue callback list, call *XrtTblAddCallback()*:

```
XrtTblAddCallback(table, XmNxrtTblCellValueCallback,
    CellValueCB, NULL);
```

Calling *XrtTblAddCallback()* is required to ensure that XRT/table knows of the existence of this callback.

You can specify whether or not the string is to be stored internally by modifying the *store* member of the callback structure. If *store* is set to `True`, the value is stored and the callback is not called again. By default, *store* is `False`, and the value is not stored.

By default, *string* can be either an XmString or a String[1]. If you want to force *string* to be of one type or the other, set the `XmNxrtTblXmStringSetMode` resource.

1. If you are linking with Motif 2.x, *string* cannot accept both XmStrings and Strings. You must use `XmNxrtTblXmStringSetMode` to specify one type or the other.

User Editing

By default, users cannot edit a cell value generated by a CellValue callback. To allow a user to do this, use a ValidateCell callback to capture the edited cell before XRT/table discards it, and store it separately. The following code can be used to take user input from a table generated by a CellValue callback and update the table:

```
static void
validateCB(Widget table, XtPointer data,
    XrtTblValidateCellCallbackStruct *cbs)
{
    if (!cbs->value_changed)
        return;
    XtFree(values[cbs->row][cbs->column]);
    values[cbs->row][cbs->column] =
                XtNewString(cbs->value);
}
```

The ValidateCell callback routine stores user cell editing in the value's structure used to populate the table. Note: `XmNxrtTblCellValueContext` is not applicable in this case.

4.6 Getting Data out of the Table

ASCII Output

The *XrtTblWriteAscii()* method creates a delimited ASCII file containing the cell values for the range of cells you specify, typically for transfer into a spreadsheet or database application. This method does not output label values. If you don't specify a range, all non-blank cells are output. For a complete description of this method's arguments, see Appendix B on page 191.

SYLK Output

The *XrtTblWriteSylk()* method creates a file in Microsoft's Symbolic Link (SYLK) format containing the cell values for the range of cells you specify. SYLK allows cell values to be imported into popular commercial spreadsheets. If you don't specify a range, *XrtTblWriteSylk()* writes all non-blank cells to the SYLK file. For a complete description of this method's arguments, see Appendix B on page 191.

5

Programming User Interaction

Default User Interaction ■ *User Interaction and Translation Customization*
List Display Mode ■ *Table Scrolling*
Row and Column Resizing ■ *Cell Traversal*
Cell Editing ■ *Cell Selection*
Mouse Pointers ■ *Drag and Drop*

This chapter describes the user-interaction features of XRT/table–how a user can interact with the table and how an application can control interaction.

5.1 Default User Interaction

By default, an end-user can scroll through the table, resize individual rows/columns, traverse from cell to cell, drag and drop text, and edit cell values. When enabled, a user can also select cells. Figure 49 shows how to perform the interactions using XRT/table's default translations.

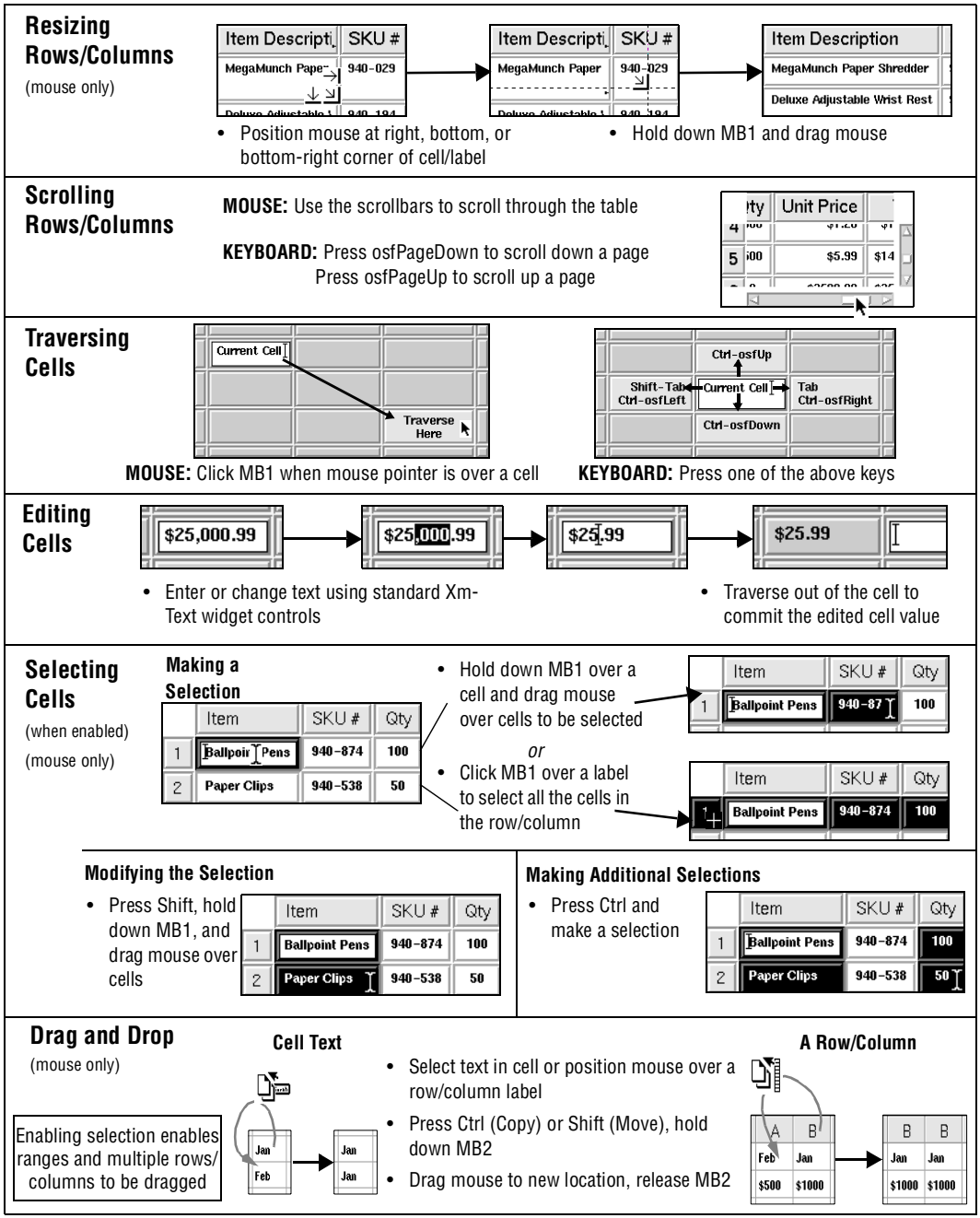


Figure 49 XRT/table's Default User Interactions

5.2 User Interaction and Translation Customization

XRT/table's default translation table defines the user events that cause some interactive action on the widget. You can customize user interaction through the following mechanisms:

- Translation tables–An application can change the default translation events or actions, or add translations for new actions to the table. This section describes how to do this. Appendix D on page 223 lists XRT/table's default translations and actions.
- Callbacks–An application can be notified as a user interacts with the widget by registering callback routines that are called before, during, and after user interaction. A callback routine can affect each interaction in such ways as disallowing or constraining it. See the section on each interaction for details on using its callback.

Setting Translations

To change an existing translation's event and/or action or add a new translation to the table, you must set the translation on the table and all of the private area-clipping widgets managed by the table (see section 2.3 on page 24 for more details on the table's child widgets). `XmNxrtTblClipChildren` specifies a list of all these widgets. The last element of the list must be set to `NULL`.

The following example adds a new translation on the table and all of its clip widget children:

```
Widget *child;
int i;
static XtActionsRec actions[] = {
    {"DisplayMenu",    (XtActionProc) DisplayMenu }
};
static char trans[] = "<Btn3Down>: DisplayMenu()";
...
XtAppAddActions(XtWidgetToApplicationContext(table),
                    actions, 1);
XtVaGetValues(table, XmNxrtTblClipChildren, &child, NULL);
XtOverrideTranslations(table,
                    XtParseTranslationTable(trans));
for (i=0; child[i]; i++)
    XtOverrideTranslations(child[i],
                    XtParseTranslationTable(trans));
```

Caution: Setting translations correctly is complex, and is complicated by bugs in some implementations of Motif. Use the following guidelines to ensure that translations you set on the table will work properly:

- Avoid using resource files to set translations.
- Do not set translations when creating the widget.
- Always use *XtOverrideTranslations()* to set translations on the table.

Disabling Translations

An easy way to disable any particular interaction is to override its default translation to call the *NoAction()* routine.

Defining New Actions

When creating an action routine, it is usually useful to know which cell or label the event was triggered on. You can determine this by calling *XrtTblEventToRowCol()*, which is fully described on page 191.

Double-clicks and Beyond

Handling two or more clicks on the table requires special coding because XRT/table uses single clicks to indicate traversal, cell resizing, selection, and editing. Each time the user clicks on a cell (within the bounds of `XmNmultiClickTime`) XRT/table updates the *click_count* member of the `XrtTblSelectCallbackStruct` structure. An application can use a Select callback (described in detail on page 102) to be notified of each click on a cell or label. The following Select callback calls a sorting procedure when the user double-clicks on a column label. Selection policy must be set to RANGE or MULTIRANGE for this example to work.

```
selectCB(Widget table, XtPointer data,
    XrtTblSelectCallbackStruct *cbs)
{

    /* Allow double-clicks */
    if (IS_COLUMNLABEL(cbs->row, cbs->column)) {
        if (cbs->reason == XRTTBL_REASON_UNSELECT) {
           if (cbs->click_count > 1)
                   cbs->doit = False;
        }

    /* Double click on a column label - sort column */
        else if (cbs->reason == XRTTBL_REASON_SELECT_END
                   && cbs->click_count == 2)
        sortCB();
    }
}
```

5.3 List Display Mode

XRT/table allows you to specify that a table is to be displayed and should operate in a manner similar to that of an XmList widget. To do this, set the `XmNxrtTblMode` resource to `XRTTBL_MODE_LIST`. Selecting a cell when the table is in list display mode will select its entire row.

Note: If the row is already selected, selecting a cell while the table is in list display mode will unselect the entire row. As with XmList, holding down the CTRL key and clicking on a selected row toggles that row's selection, but does not affect other selected rows.

To mimic the behavior and appearance of an XmList widget, do the following:

- Specify that all cells are to be non-traversable by setting `XmNxrtTblTraversableContext` to `False` for the context (`ALL, ALL`).
- Remove all cell borders by setting `XmNxrtTblBorderTypeContext` to `XRTTBL_BORDER_NONE`.

- Set the XmNxrtTblSelectionPolicy resource to XRTTBL_SELECT_SINGLE. This ensures that the user can only select one row at a time.

By default, XmNxrtTblMode is set to XRTTBL_MODE_TABLE, which indicates that the table is to operate normally.

5.4 Table Scrolling

When a table is larger than the rows/columns visible on the screen, an end-user can scroll through the table with the mouse or keyboard. To provide this ability, XRT/table creates and controls two XmScrollBar widgets (one horizontal, one vertical) for mouse scrolling, and uses the *Scroll()* action routine for keyboard scrolling.

The XmNxrtTblTopRow and XmNxrtTblLeftColumn resources are updated dynamically as a table is scrolled.

XRT/table also scrolls the table when necessary as part of other interactions, such as cell traversal and cell selection. Scrolling does not change the location of the current cell.

5.4.1 Scrollbars

An application can control how scrollbars are attached to the widget, when they are displayed, and the side they are located on. The behavior of the scrollbars themselves can also be controlled.

Scrollbar Attachment

The way scrollbars should be attached to the table depends on the style of table you need for your application. *Standard-style* tables attach the scrollbars to the cell/label area and move them to match any changes to the size of the visible area. *XmScrolledList-style* tables attach the scrollbars to the table widget itself and do not move when the size of the visible area changes–only changes in the widget size cause the scrollbars to move and change size.

The following three resources specify the attachment of the horizontal scrollbar: XmNxrtTblHorizScrollBarPosition, XmNxrtTblHorizScrollBarLeftAttachment, and XmNxrtTblHorizScrollBarRightAttachment.

The following three resources specify the attachment of the vertical scrollbar: XmNxrtTblVertScrollBarPosition, XmNxrtTblVertScrollBarTopAttachment, and XmNxrtTblVertScrollBarBottomAttachment.

To specify standard-style table scrollbars:

- Leave the Position and Attachment resources at their default values.

To specify XmScrolledList-style table scrollbars:

- Set the horizontal and vertical Position resources to XRTTBL_SBPOSITION_SIDE.
- Set the left, right, top, and bottom Attachment resources to XRTTBL_ATTACH_SIDE.

- Set XmNxrtTblDisplayHorizScrollBar and XmNxrtTblDisplayVertScrollBar to XRTTBL_DISPSB_ALWAYS.
- Set XmNxrtTblSpace to 0.

XmNxrtTblSpace specifies the offset between the scrollbars and the table. This offset usually applies between the scrollbars and the table's cells/labels. However, when the scrollbars are attached to the side of the widget, it *can* apply between the scrollbars and the side of the widget (only when there is space between the last row/column and the edge of the widget).

Scrollbar Display

By default, XRT/table displays each scrollbar only when the table is larger than the number of rows/columns visible on the screen. To display a scrollbar at all times, set XmNxrtTblDisplayHorizScrollBar and/or XmNxrtTblDisplayVertScrollBar to XRTTBL_DISPSB_ALWAYS. Set them to XRTTBL_DISPSB_NEVER to completely disable scrollbar display.[1]

Scrollbar Placement

Each scrollbar can be placed on one side of the table or the other using the XmNxrtTblHorizScrollBarPlacement and XmNxrtTblVertScrollBarPlacement resources. Figure 50 illustrates the effect of switching the placement of both scrollbars.

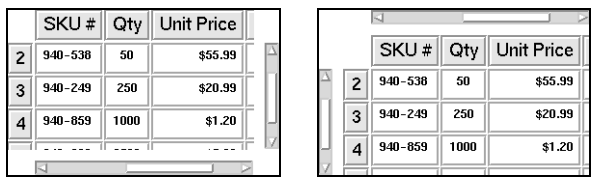


Figure 50 Default and opposite scrollbar placement

Jump Scrolling

The scrollbars' scrolling behavior is specified by XmNxrtTblJumpScroll. By default (XRTTBL_JUMP_VERTICAL), the vertical scrollbar "jump-scrolls" the table one whole row at a time. To jump-scroll with both scrollbars, set this resource to XRTTBL_JUMP_ALL. To cause the scrollbars to scroll smoothly through the table, set it to XRTTBL_JUMP_NONE. You can cause only the horizontal scrollbar to jump-scroll by setting this resource to XRTTBL_JUMP_HORIZONTAL.

Preview Scrolling

When XmNxrtTblPreviewScrolling is active, a small window displays next to the scrollbar thumb when the thumb is dragged (clicked and held). This window displays the row/column that would be displayed at the top-left of the table if the thumb were released.

1. Although scrollbars are removed, a user can still scroll with the keyboard. See "Disabling Interactive Scrolling" for complete information on disabling interactive scrolling.

The window's appearance can be configured using the *XrtTblSetPreviewProperties()* and *XrtTblGetPreviewProperties()* methods. Items that can be configured include the window's foreground/background colors, the window's border and the font used.

XmScrollBar Attributes

You can set or retrieve any of the resources of either scrollbar. `XmNxrtTblHorizontalScrollBar` specifies the widget ID of the horizontal scrollbar; `XmNxrtTblVerticalScrollBar` specifies the widget ID of the vertical scrollbar. The following code sets the background color of the vertical scrollbar:

```
Widget scrollbar;
XtVaGetValues(table,
    XmNxrtTblVerticalScrollbar, &scrollbar,
    NULL);
XtVaSetValues(scrollbar,
    XtVaTypedArg, XmNbackground,
    XmRString, "red", strlen("red") + 1,
    NULL);
```

Resources of the scrollbars may also be set in a resources file. The scrollbars are named "`vertScroll`" and "`horizScroll`". To address the horizontal scrollbar of all the tables in your application, you could use "`*.XtXrtTable.horizScroll`". Consult Appendix G on page 257 for full details on specifying resources in resource files.

Consult your Motif documentation for more information on programming the XmScrollBar widgets.

5.4.2 Programming Scrolling

Disabling Interactive Scrolling

Scrolling can be disabled in one or both directions. Mouse and keyboard scrolling must be disabled separately:

- Mouse–remove the scrollbars from the screen by setting `XmNxrtTblDisplayHorizScrollBar` and/or `XmNxrtTblDisplayVertScrollBar` to `XRTTTBL_DISPSB_NEVER`.
- Keyboard–override the scrolling translations to call the *NoAction()* routine, as shown by the following code:

```
*.XtXrtTable.translations: #override \n\
    <Key> osfPageDown:  NoAction()
    <Key> osfPageUp:    NoAction()
```

To fully disable any and all scrolling, an application should also ensure that the user cannot select cells or traverse to cells outside the visible area.

Forcing Scrolling

An application can force the table to scroll in the following ways:

- To scroll a particular row to the top of the display, set `XmNxrtTblTopRow` to the row. To scroll a particular column to the left side of the display, set `XmNxrtTblLeftColumn` to the column.
- To determine whether a cell or label is already visible, call *XrtTblIsCellVisible()*.

- To scroll to display a particular cell, call *XrtTblMakeCellVisible()*.

5.4.3 Scroll Callbacks

XRT/table provides a way for your application to be notified when the table is scrolled by either the end-user or the application. The `XmNxrtTblScrollCallback` resource allows you to define a list of procedures to be called when the table scrolls; this is useful if your application is drawing into the table. Each callback in the list is passed the following structure:

```
typedef struct {
    int             reason;              // Read-only
    XEvent          *event;              // Read-only
    unsigned char   direction;
    int             new value;
    String          preview_text;        //Read-Write
    XmScrollBarCallbackStruct *sb_cbs;   // Read-only
} XmNxrtTblScrollCallback;
```

The Scroll callbacks are called both before and after each scrollbar event occurs. (If the user is dragging the scrollbar "thumb", many such events may occur.) The *reason* member is either `XRTTBL_REASON_SCROLL_BEGIN` (before cells are redrawn) or `XRTTBL_REASON_SCROLL_END` (after cells are redrawn). The *new value* is the new value of the `XmNxrtTblTopRow` or `XmNxrtTblLeftColumn` resource. *preview_text* is the text that would display in the table if the thumb were released. *XtFree()* the old value and allocate memory for the new value before replacing it. XRT/table frees the memory used by this element of the callback structure.

The *sb_cbs* member contains the callback structure passed to the XmScrollBar widget itself. This contains the value of the slider's new location, and the place where selection occurred.

5.5 Row and Column Resizing

The *ResizeCell()* action allows a user to interactively resize a row and/or column (when allowed by `XmNxrtTblAllowCellResize`). This action routine alters `XmNxrtTblPixelHeight` when resizing rows, and `XmNxrtTblPixelWidth` when resizing columns. Users cannot resize rows or columns to smaller than 5 pixels.

Disallowing Cell Resizing

Use the `XmNxrtTblAllowCellResize` resource to control interactive row/column resizing over the entire table. No row/column resizing is allowed when it is set to `XRTTBL_RESIZE_NONE`. Only columns may be resized when it is set to `XRTTBL_RESIZE_HORIZONTAL`. Only rows may be resized when it is set to `XRTTBL_RESIZE_VERTICAL`.

You can specify that certain rows or columns are to become flexible by setting `XmNxrtTblPixelHeightContext` or `XmNxrtTblPixelWidthContext`. See section 3.7.4 on page 51 for more information on flexible rows and columns.

Controlling Resizing

You can use a ResizeCell callback (specified with `XmNxrtTblResizeCellCallback`) to control interactive row/column resizing on a case-by-case basis. Each callback is passed a pointer to the following structure:

```
typedef struct {
    XrtTblReason    reason;                 // Read-only
    XEvent          *event;                 // Read-only
    String          *params;                // Read-only
    Cardinal        num_params;             // Read-only
    int             row, column;            // Read-only
    int             current_row_height,     // Read-only
    int             current_column_width,   // Read-only
    int             new_row_height,
    int             new_column_width,
    Boolean         doit;
} XrtTblResizeCellCallbackStruct;
```

ResizeCell callbacks are called before resizing begins (*reason* is `XRTTBL_REASON_RESIZE_BEGIN`), after a user has selected a new row height/column width (*reason* is `XRTTBL_REASON_RESIZE_END`), and after the table has resized the row/column (*reason* is `XRTTBL_REASON_RESIZE_DONE`). The following shows the members a callback can change at each stage:

Reason	Changeable Members	Use in Callback
XRTTBL_REASON_RESIZE_BEGIN	doit	Change to `False` to disallow the user resize
XRTTBL_REASON_RESIZE_END	new_row_height new_column_width	Change to new values to constrain the resize
XRTTBL_REASON_RESIZE_DONE	None—notification only	

The *ResizeCell()* action applies these members For example, to disallow resizing of row 5 only, the callback would change *doit* to `False` when *reason* is `XRTTBL_REASON_RESIZE_BEGIN` and *row* is 5. The following example callback routine sets the width of any resized column to an increment of 10 pixels:

```
static void
cellresizeCB(Widget table, XtPointer data,
    XrtTblResizeCellCallbackStruct *cbs)
{
    switch (cbs->reason) {
    case XRTTBL_REASON_RESIZE_BEGIN:
        break;
    case XRTTBL_REASON_RESIZE_END:
        cbs->new_column_width = \
            cbs->new_column_width / 10 * 10;
        break;
    case XRTTBL_REASON_RESIZE_DONE:
        break;
    }
}
```

To register the above callback routine, use the following call:

```
XtAddCallback(table, XmNxrtTblResizeCellCallback,
        cellresizeCB, NULL);
```

5.6 Cell Traversal

Traversal is the act of moving the current cell from one location to another. The *TraverseToCell()* action performs interactive cell traversal. A traversal passes through three stages: validating the edited current cell, determining the new current cell location, and entering that cell. Figure 51 illustrates these stages, and the sequence that callbacks are triggered.

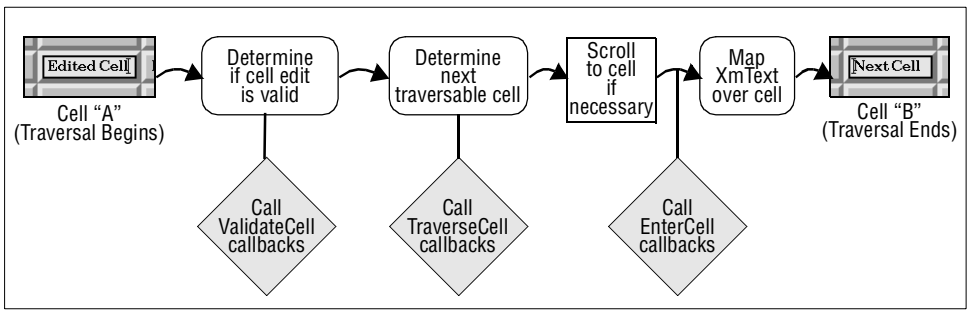


Figure 51 Sequence of cell traversal callbacks

The cell validation and cell entry stages are detailed processes of their own, and are discussed separately. See section 6.2 on page 116 for details on cell validation. See section 5.7 on page 96 for details on cell entry prior to editing.

Note that when users are allowed to select cells (specified with `XmNxrtTblSelectionPolicy`), a traversal internally triggers Select callbacks.

Disallowing Cell Traversal

By default, all cells are traversable.[1] To prevent users from traversing to a cell, set `XmNxrtTblTraversableContext` to `False`. Making a cell non-traversable also prevents it from being traversed to programmatically (using *XrtTblTraverseToCell()* or a TraverseCell callback).

Disabling traversal also disables cell editability regardless of the setting of `XmNxrtTblEditableContext`.

The following example sets all cells in row 3 to be non-traversable:

```
XtVaSetValues(table,
    XmNxrtTblContext, XrtTblSetContext(3, XRTTBL_ALLCELLS),
    XmNxrtTblTraversableContext,  False,
    NULL);
```

1. Cells smaller than 3 pixels cannot be traversed to, whether or not they are traversable.

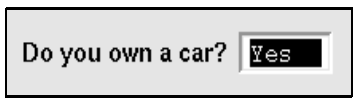
Minimum Cell Visibility

By default, when a user traverses to a cell that is not currently visible, XRT/table scrolls the table to display the entire cell. You can control how much of the cell is scrolled into view using `XmNxrtTblMinCellVisibility`.

Pre-selecting the Text

You can specify that the cell text should be selected when traversed to. To do this, set `XmNxrtTblTextInitSelect` to `True`. By default, cell text is not marked as selected.

This behavior is common in forms: the user can enter a new cell value without having to delete the old one first.



If the user types a character, `Yes` is deleted and replaced by the user's text. If the user traverses out of the cell, `Yes` remains in the cell.

Figure 52 A cell containing pre-selected text

Forcing Traversal

An application can force the current cell to traverse to a particular cell by calling *XrtTblTraverseToCell()*. If the cell is non-traversable (specified with `XmNxrtTblTraversableContext`), this method returns `False`. *XrtTblTraverseToCell()* is fully described in Appendix B on page 191.

Controlling Interactive Traversal

You can use a TraverseCell callback (specified with `XmNxrtTblTraverseCellCallback`) to control interactive traversal. Each callback is passed a pointer to the following structure:

```
typedef struct {
    XrtTblReason    reason;         // Read-only
    XEvent          *event;         // Read-only
    String          *params;        // Read-only
    Cardinal        num_params;     // Read-only
    int             row, column;    // Read-only
    int             next_row;
    int             next_column;
} XrtTblTraverseCellCallbackStruct;
```

A TraverseCell callback can change the *next_row* and *next_column* members. The *TraverseToCell()* action attempts to traverse to the cell specified by these members. Note that if *next_row* and *next_column* are changed to a non-traversable cell, the traversal will not be successful. The following example callback prevents the user from traversing outside of column 0:

```
static void
traverseCB(Widget table, XtPointer data,
    XrtTblTraverseCellCallbackStruct *cbs)
{
    if (cbs->next_column > 0) {
        cbs->next_row = cbs->row + 1;
        cbs->next_column = 0;
    }
}
```

Positioning the Text Cursor

When the user selects a cell for editing, the cursor is normally placed at the beginning of the first line of text in the cell. To place the cursor at the point where the mouse is clicked, set the `XmNxrtTblSetTextCursorPos` resource to `True`.

5.7 Cell Editing

XRT/table creates and controls one XmText widget to provide end-users with the ability to edit the value of a cell. By default, a user can edit any cell. When a user traverses to a cell, the XmText widget is mapped over the cell, containing a copy of the cell value (from the `XrtTblCellValues` structure). Any change made by the user is written to the `XrtTblCellValues` structure when the user traverses to another cell.

An application can control the following aspects of interactive cell editing on a cell-by-cell basis: editability, case, multi-line support, combo/spin box support and maximum text length. An application can also control attributes and behavior of the current cell itself by programming the table's XmText widget.

Disallowing Cell Editing

To prevent users from editing a cell value, set `XmNxrtTblEditableContext` to `False`. Unless traversal has been disabled (specified by `XmNxrtTblTraversableContext`), a user can traverse to a non-editable cell and copy the value using the standard X copy mechanism.

Note that cells that you want a user to be able to edit must also be traversable (`XmNxrtTblTraversableContext` must be `True`).

The following example sets row 5 non-editable (but traversable):

```
XtVaSetValues(table,
    XmNxrtTblContext, XrtTblSetContext(5, XRTTBL_ALLCELLS),
    XmNxrtTblEditableContext,    False,
    XmNxrtTblTraversableContext, True,
    NULL);
```

Case Conversion

By default, the case of text entered by a user is not changed. Use `XmNxrtTblCaseContext` to ensure that the text of any cell traversed to or edited by a user is all upper case (`XRTTBL_CASE_UPPER`) or all lower case (`XRTTBL_CASE_LOWER`). This resource takes effect from the point it is set; it does not convert the case of previously-entered cell values.

Multiple-Line Cells

By default, a user can only enter a single line of text into a cell. To allow a user to enter multiple lines of text, set `XmNxrtTblMultilineContext` to `True`. This resource sets the corresponding `XmNeditMode` XmText resource when the cell is traversed to. If the cell already contains a multiline value, this resource is ignored.

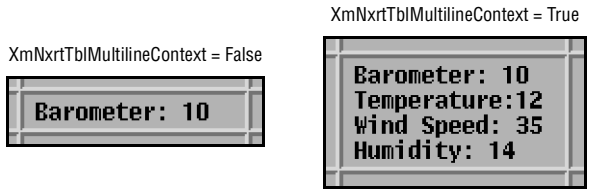


Figure 53 A Single and Multi-Line cell

Using Combo Boxes and Spin Boxes

Using the `...PickList...` resources and `XmNxrtTblDataTypeContext`, a user can be presented with a combo box or spin box to facilitate editing the table cell. Note that this requires XRT/field. See "Using XRT/field with XRT/table" on page 119.

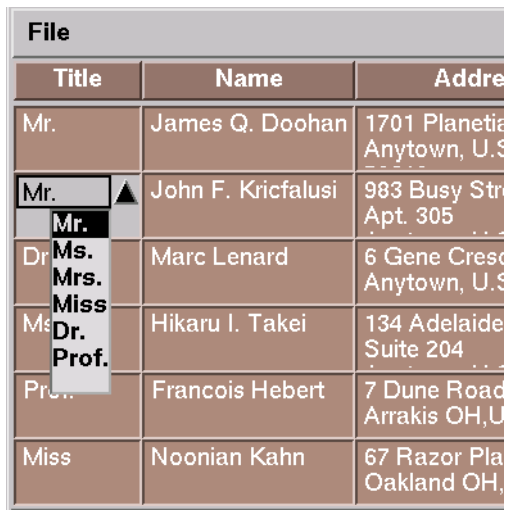


Figure 54An opened combo box within a cell

Text Length

To limit the number of characters a user can enter in a cell, set `XmNxrtTblMaxLengthContext` to the number of characters. This resource sets the corresponding `XmNmaxLength` XmText resource when the cell is traversed to. Note that this maximum includes newline characters in multiline cells.

XmText Widget Attributes

You can set or retrieve any of the XmText resources of the current cell. `XmNxrtTblText` specifies the widget ID of the current cell. The following code sets the background color of the current cell:

```
Widget current_cell;
XtVaGetValues(table,
    XmNxrtTblText, &current_cell,
    NULL);
XtVaSetValues(current_cell,
    XtVaTypedArg, XmNbackground,
    XmRString, "grey80", strlen("grey80") + 1,
    NULL);
```

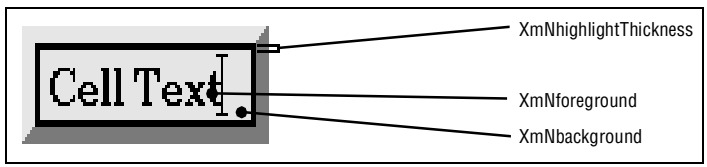


Figure 55 Current Cell XmText attributes

Resources may be set for XmText widgets using the name "`.text`". Consult Appendix G on page 257 for full details on specifying resources in resource files.

Consult your Motif documentation for more information on programming the XmText widget.

Controlling Editing at Runtime

You can use an EnterCell callback (specified with `XmNxrtTblEnterCellCallback`) to control user cell editing at runtime. Triggered by the *TraverseToCell()* action, EnterCell callbacks are called at the end of a traversal, as illustrated by Figure 51. Each callback is passed a pointer to the following structure:

```
typedef struct {
    XrtTblReason        reason;         // Read-only
    XEvent              *event;         // Read-only
    String              *params;        // Read-only
    Cardinal            num_params;     // Read-only
    int                 row, column;    // Read-only
    Boolean             doit;
} XrtTblEnterCellCallbackStruct;
```

EnterCell callbacks are called before (*reason* is `XRTTBL_REASON_ENTER_CELL_BEGIN`) and after (*reason* is `XRTTBL_REASON_ENTER_CELL_END`) completing traversal to a cell. The following shows the members a callback can change at each stage:

Reason	Changeable Member	Use in Callback
XRTTBL_REASON_ENTER_CELL_BEGIN	doit	Change to `False` to disallow editing of the cell by the user
XRTTBL_REASON_ENTER_CELL_END	None—notification only	

For example, to disallow editing of a particular cell based on its value, the callback would (when *reason* is `XRTTBL_REASON_ENTER_CELL_BEGIN`) get the cell value and change *doit* to `False` if editing that particular value should not be allowed. The following callback routine sets the cell value to the current time when a user traverses to it:

```
static void
enterCB(Widget table, XtPointer data,
    XrtTblEnterCellCallbackStruct *cbs)
{
    time_t now;

    if (cbs->reason != XRTTBL_REASON_ENTER_CELL_BEGIN)
        return;
    now = time(0);
    XtVaSetValues(table,
        XmNxrtTblContext,
                XrtTblSetContext(cbs->row, cbs->column),
        XmNxrtTblCellValueContext,  ctime(&now),
        NULL);
}
```

5.8 Cell Selection

When enabled, XRT/table allows a user to interactively select one or more ranges of cells. An application can retrieve each range to manipulate the cells within it. An application can also be notified of each user selection to control what and how the user selects cells.

Selection Policy

The `XmNxrtTblSelectionPolicy` resource controls the amount of selection allowed on the table, both by end-users, and by the application. By default, no selection is allowed. Changing the selection policy affects subsequent selection attempts; it does not affect current selections. Figure 56 shows the valid values, and the amount of selection they allow.

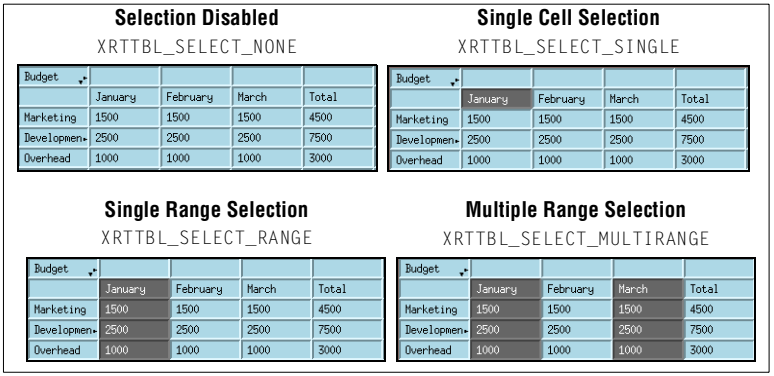


Figure 56 Selection policies

Selected Cell List

The `XmNxrtTblSelectedCellList` resource specifies the list of all currently selected ranges in the table; the last element of this list must be set to `NULL`. This resource is updated dynamically as a user selects cells. It is also updated when an application programmatically selects or deselects cells. Labels cannot be part of a selected range.[1]

Each range in the selected cell list is an `XrtTblRange` structure:

```
typedef struct {
    int             start_row, start_column;
    int             end_row, end_column;
} XrtTblRange;
```

All members of the `XrtTblRange` structure can be a row and column index. *end_row* and *end_column* can also be `XRTTBL_MAXINT` to specify all of the cells in a row or column. Because the user can select in any direction, the start point is not necessarily the top-left corner of the range–it may be any of the four corners of a range.

The following example sets two selected ranges:

```
static XrtTblRange selection1 = {0, 0, 4, 2};
static XrtTblRange selection2 = {7, 1, 7, 4};
static XrtTblRange *all_selections[] = {
    {&selection1, &selection2, NULL};

XtVaSetValues(table,
    XmNxrtTblSelectionPolicy, XRTTBL_SELECT_MULTIRANGE,
    XmNxrtTblSelectedCellList, all_selections,
    NULL);
```

1. Clicking on a label selects all of the cells in the row or column, not including the label.

Selection Colors

You can customize the colors used to indicate selected cells through the `XmNxrtTblSelectedBackground` and `XmNxrtTblSelectedForeground` resources. See section 3.9 on page 53 for more details.

Working with Selected Ranges

To get a selected range, allocate an `XrtTblRange` structure and call *XrtTblGetSelectedRange()*. This method has the following prototype:

```
Boolean
XrtTblGetSelectedRange(
    Widget      table,
    int         position
    XrtTblRange *range              /* Returned */
    )
```

The following example gets each selected range (assumes that selection policy is `XRTTBL_SELECT_MULTIRANGE`):

```
XrtTblRange range;
int i = 0;
while (XrtTblGetSelectedRange(table, i, &range)) {
    /* Process this range */
...
    i++;
}
```

To determine whether a particular cell is selected, and retrieve the range if it is, call *XrtTblIsCellSelected()*. This method has the following prototype:

```
Boolean
XrtTblIsCellSelected(
    Widget       table,
    int          row, column
    int          *range_index,   /* Returned */
    XrtTblRange  *range          /* Returned */
    )
```

Forcing Selection

An application can add a selection to the selected cell list by defining the range, calling *XrtTblRangeListAdd()* to add the range to a copy of the currently-selected ranges, and then setting `XmNxrtTblSelectedCellList` to the new range list, as shown by the following example:

```
XrtTblRange **all_selections;
static XrtTblRange myrange = {0, 0, 4, 2};

XtVaGetValues(table, XmNxrtTblSelectedCellList,
                    &all_selections, NULL);
all_selections = XrtTblRangeListCopy(all_selections);
all_selections = XrtTblRangeListAdd(all_selections,
                    XRTTBL_MAXINT, &myrange);

XtVaSetValues(table,
    XmNxrtTblSelectionPolicy,   XRTTBL_SELECT_MULTIRANGE
    XmNxrtTblSelectedCellList,      all_selections,
    NULL);
XrtTblRangeListFree(all_selections);
```

Removing Selections

To remove a range from the current list, use *XrtTblRangeListDelete()* on a copy of the selected cell list and set `XmNxrtTblSelectedCellList` to the new range list.

To remove all selections from the table, set `XmNxrtTblSelectedCellList` to `NULL`.

To find a specific row/column index within a list of ranges, use *XrtTblRangeListFind()*. To find a specific range in a list of ranges, use *XrtTblRangeListFindRange()*. In either case, if no match is found, `XRTTBL_NOTFOUND` is returned.

Runtime Selection Control

You can use a Select callback (specified with `XmNxrtTblSelectCallback`) to control interactive cell selection at each stage, on a case-by-case basis. Each callback is passed a pointer to the following structure:

```
typedef struct {
    XrtTblReason    reason;         // Read-only
    XEvent          *event;         // Read-only
    String          *params;        // Read-only
    Cardinal        num_params;     // Read-only
    int             range_index;    // Read-only
    XrtTblRange     *range;         // Read-only
    int             row, column;
    int             click_count;
    Boolean         doit;
} XrtTblSelectCallbackStruct;
```

Select callbacks are called before selection begins (*reason* is `XRTTBL_REASON_SELECT_BEGIN`), before deselecting any selected cells (*reason* is `XRTTBL_REASON_UNSELECT`), as each cell is selected (*reason* is `XRTTBL_REASON_SELECT_EXTEND`), and after a selection has finished (*reason* is `XRTTBL_REASON_SELECT_END`). The following shows the members a callback can change at each stage:

Reason	Changeable Members	Use in Callback
XRTTBL_REASON_SELECT_BEGIN	row, column doit	Change row/column to start selection at a new location. Change doit to `False` to disallow user selection.
XRTTBL_REASON_UNSELECT	doit	Change to `False` to disallow user deselection.
XRTTBL_REASON_SELECT_EXTEND	row, column doit	Change row/column to constrain user selection. Change doit to `False` to disallow selection.
XRTTBL_REASON_SELECT_END	None—notification only	

The *Select()* action applies these members during interactive selection. For example, to disallow selection of a particular cell, a callback would set the *doit* member to `False` when the *reason* member is `XRTTBL_REASON_SELECT_BEGIN` or `XRTTBL_REASON_SELECT_EXTEND`. The following example callback routine constrains selection to the column it was started:

```
static void
selectCB(Widget w, XtPointer data,
    XrtTblSelectCallbackStruct *cbs)
{
    static int column;
    if (cbs->reason == XRTTBL_REASON_SELECT_BEGIN) {
        column = cbs->column;
    }
    else if (cbs->reason == XRTTBL_REASON_SELECT_EXTEND) {
        cbs->column = column;
    }
}
```

Note: When users are allowed to select cells, *cell traversal* internally triggers Select callbacks. Check the *event* member of the Select callback structure if you want your callback to work with user events only–when *event* is `NULL`, the callback is internally triggered by the table.

Other Select Callback Uses

There are several other ways to use this callback, including:

- Entire Row/Column selection–change the *row/column* member to the row/column label when *reason* is `XRTTBL_REASON_SELECT_BEGIN` to allow a user to select an entire row/column by clicking in it.
- Disallowing deselection–change *doit* to False when *reason* is `XRTTBL_REASON_SELECT_BEGIN` to disallow deselecting cells by a user.
- Double-clicks–examine the *click_count* member to determine the number of times the user has clicked on a cell/label within the time specified by `XmNmultiClickTime`. This is described in detail in section 5.2 on page 87.

Cell Sensitivity

By using `XmNxrtTblSensitiveContext`, you can specify whether a cell is context sensitive or not. A cell which is not sensitive is drawn grayed-out, but it can still be selected and traversed into. Other cell sensitivity points to note are:

- `XmNxrtTblSensitiveSeries` can be specified in a resource file with the same syntax as other series resources.
- This resource is available in the `ContextValueCallback`. See "Managing Contexts Programmatically" on page 139.
- This resource is applied to widgets in cells, cloned or not. See "Widgets in a Table" on page 110.
- If the entire table is insensitive, this resource has no effect.
- You can drag and drop onto an insensitive cell.

5.9 Mouse Pointers

XRT/table tracks the mouse pointer as a user moves over and interacts with the widget. When the pointer is over cells or labels, it changes shape to provide users with a visual cue as to the interaction available at that point. The following lists XRT/table's default pointers:

Mouse Pointer	Interaction Cue	Resource
I-beam Pointer  `XC_xterm`	■ Cell under pointer is editable	`XmNxrtTblCursorTraversable`
Cross-hair Pointer  `XC_plus`	■ Cell or label under pointer is non-traversable[a]	`XmNxrtTblCursorNontraversable`
Column Resize Pointer  `XC_right_side`	■ Column left of pointer can be resized	`XmNxrtTblCursorVertical`
Row Resize Pointer  `XC_bottom_side`	■ Row above pointer can be resized	`XmNxrtTblCursorHorizontal`
Row/Column Resize Pointer  `XC_bottom_right_corner`	■ Row above and column left of pointer can be resized	`XmNxrtTblCursorCorner`

a. Always appears over labels because they can never be edited by users

When tracking the mouse pointer, XRT/table considers the current settings of the `XmNxrtTblTraversableContext` and `XmNxrtTblAllowCellResize` resources. For example, if row resizing has been disabled, the ⇩ and ⇲ pointers will not display.

Changing Default Pointers

An application can change these default pointers using the resources listed in the above table. The *XCreateFontCursor()* Xlib procedure can be used to create a mouse pointer using the standard cursor font. The returned cursor can be used as a value for XRT/table's cursor resources. Consult your Xlib documentation for details on creating mouse cursors.

To set all the cursors in XRT/table to a specific cursor, use *XrtTblSetAllCursors()*. To restore all the cursors in XRT/table to their default values, use *XrtTblRestoreAllCursors()*.

Disabling Pointer Tracking

To use an application-defined mouse pointer over the entire widget, set `XmNxrtTblTrackCursor` to `False`; XRT/table will not track the position of the mouse over the widget.

TrackCursor Callback

An application can use a TrackCursor callback (specified with `XmNxrtTblTrackCursorCallback`) to be notified as the user moves the mouse over the table. Each callback is passed a pointer to the following structure:

```
typedef struct {
    XrtTblReason        reason;         // Read-only
    XEvent              *event;         // Read-only
    int                 row, column;    // Read-only
    XrtTblBorderSides   sides;          // Read-only
    Boolean             enter_cell;     // Read-only
    Cursor              old_cursor;     // Read-only
    Cursor              new_cursor;
} XrtTblTrackCursorCallbackStruct;
```

TrackCursor callbacks are called when the pointer is leaving a cell/label (*reason* is `XRTTBL_REASON_CURSOR_LEAVE`), on the border between cells/labels (*reason* is `XRTTBL_REASON_CURSOR_EDGE`), and when the pointer is inside a cell/label (*reason* is `XRTTBL_REASON_CURSOR_INSIDE`).

A callback can change the *new_cursor* member to specify a different mouse pointer at runtime. This is useful for disabling display of resize pointers when over rows/columns that have resizing disallowed using a ResizeCell callback, as shown by the following example:

```
/* Disallow resize of row 3 */
static void
resizeCB(Widget table, XtPointer data, XrtTblResizeCellCallbackStruct *cbs)
{
    if (cbs->row == 2)
            cbs->doit = False;
}

/* Change mouse pointer when over bottom of row 3 or top of row 4 */
static void
trackCB(Widget table, XtPointer data, XrtTblTrackCursorCallbackStruct *cbs)
{
    Cursor cursor;

    if ((cbs->row == 2
            && cbs->reason == XRTTBL_REASON_CURSOR_EDGE
            && (cbs->sides & XRTTBL_BORDERSIDE_BOTTOM) != 0)
        || (cbs->row == 3
            && cbs->reason == XRTTBL_REASON_CURSOR_EDGE
            && (cbs->sides & XRTTBL_BORDERSIDE_TOP) != 0)) {
        XtVaGetValues(table, XmNxrtTblCursorTraversable, &cursor, NULL);
        cbs->new_cursor = cursor;
    }
}

...
XtAddCallback(table, XmNxrtTblTrackCursorCallback, trackCB, NULL);
XtAddCallback(table, XmNxrtTblResizeCellCallback, resizeCB, NULL);
```

5.10 Drag and Drop

XRT/table supports the drag and drop mechanism introduced in Motif 1.2. Drag and drop provides a way to interactively transfer text within and between applications. Text can be either moved or copied.

By default, only cell text is affected by a drag and drop operation. Unless explicitly specified or unless an entire column or row are moved, row and column labels are not moved.

Drag and drop capabilities can be modified and extended by your application; see section 6.9 on page 130 for complete information.

Specifying a Selection Policy

By default, text from a single cell or one entire row/column can be dragged and dropped. A cell range and multiple rows/columns can be dragged and dropped when `XmNxrtTblSelectionPolicy` is set to `XRTTBL_SELECT_RANGE` or `XRTTBL_SELECT_MULTIRANGE`. Figure 57 illustrates copying and moving a range of cells with drag and drop.

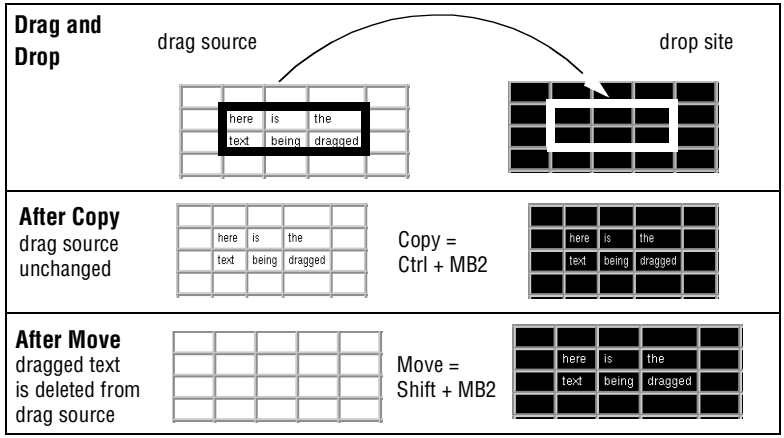


Figure 57 Dragging and dropping a range of cells

Pressing the middle mouse button without pressing Ctrl or Shift produces the following behavior:

- Cells and ranges of cells are copied.
- Entire rows and columns are moved.

Disabling Drag and Drop

To disable all drag operations in a table, override its default translations to call the *NoAction()* action routine, as shown by the following code:

```
static char trans[] = { "Shift <Btn2Down>:  NoAction()",
                        "Ctrl <Btn2Down>:   NoAction()",
                        "<Btn2Down>:        NoAction()" };
...
XtOverrideTranslations(table,
                   XtParseTranslationTable(trans));
for (i=0; child[i]; i++)
    XtOverrideTranslations(child[i],
                   XtParseTranslationTable(trans));
```

Note: these translations cannot be set in a resource file.

To disable drop, set `XmNxrtTblImportTargets` to `NULL`. To suppress only dragging from a table, set the `XmNdragInitiatorProtocolStyle` resource to `XmDRAG_NONE`. To suppress only dropping into a table, set `XmNdragReceiverProtocolStyle` to `XmDRAG_NONE`.

Note that dragging and dropping a single traversable cell uses the XmText widget's drag and drop mechanism. To change the drag and drop behavior of single traversable cells, you must override the XmText drag and drop translations.[1]

1. By default, XmText's drag and drop behavior is identical to XRT/table's.

Disabling Row/Column Labels Dragging

By default, when dragging entire rows or columns, XRT/table drags their labels. To disable dragging of row or column labels, remove the `XRTTBL_TGT_ROWLABELS` or the `XRTTBL_TGT_COLUMNLABELS` target from the list specified by `XmNxrtTblImportTargets`. The following example disables row and column label drag and drop:

```
Widget dragtable, droptable;
XrtTblImportExportStruct **implist, **explist;

/* disable row label dragging */
explist =
    XrtTblImportExportListCopy(defaultExportTargets);
explist = XrtTblImportExportListDelete(explist,
    XRTTBL_TGT_ROWLABELS);
XtVaSetValues(dragtable,
    XmNxrtTblExportTargets, explist,
    NULL);
XrtTblImportExportListDestroy(explist);

/* disable row label dropping */
implist =
    XrtTblCopyImportExportList(defaultImportTargets);
implist = XrtTblImportExportListDelete(implist,
    XRTTBL_TGT_ROWLABELS);
XtVaSetValues(droptable,
    XmNxrtTblImportTargets, implist,
    NULL);
XrtTblImportExportListDestroy(implist);
```

Suppressing Individual Cells

To suppress dropping into specified individual cells, set `XmNxrtTblEditableContext` to `False` for each cell (or range) to which dropping is to be disallowed:

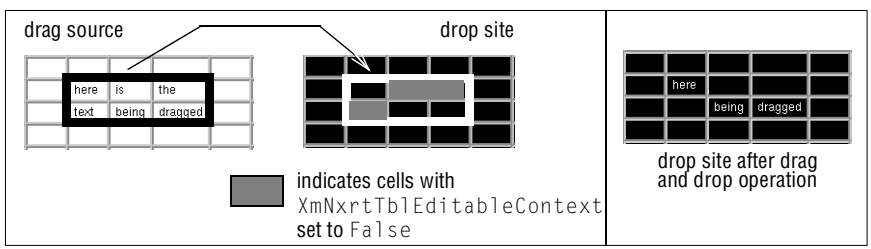


Figure 58 Suppressing dropping into individual cells

To suppress dragging and dropping of entire rows or columns, set `XmNxrtTblEditableContext` to `False` with the context `(ALL, LABEL)` for row labels or `(LABEL, ALL)` for column labels. This works even if your table does not have row labels or column labels defined.

If you are moving an entire row or column, you can move non-editable cells only if the row or column label of the drop site is editable.

6

Advanced XRT/table Programming

Widgets in Cells/Labels ■ *Cell/Label Value Validation* ■ *Using XRT/field with XRT/table*
Table Sizing and Resizing ■ *Compound Strings* ■ *Series Resource Programming*
Multiple Tables with Common Cell Values ■ *XRT/graph and XRT/3d Data*
Programming Drag and Drop ■ *XRT/table Internationalization* ■ *Performance Improvement*
Warning and Error Message Customization ■ *Using C++ with XRT/table*
Using UIL with XRT/table ■ *Reading and Writing Resource Files*

This chapter covers topics that programmers of advanced XRT/table applications will find useful. It assumes that you are already familiar with XRT/table.

6.1 Widgets in Cells/Labels

Any cell or label can contain any Motif widget as well as text and a pixmap. XRT/table can display Primitive, Composite, and third-party widgets. Widgets in cells/labels are fully functional–you can program their attributes and handle user interaction with them. The table manages the widget's sizing, geometry, and some attributes (such as color).

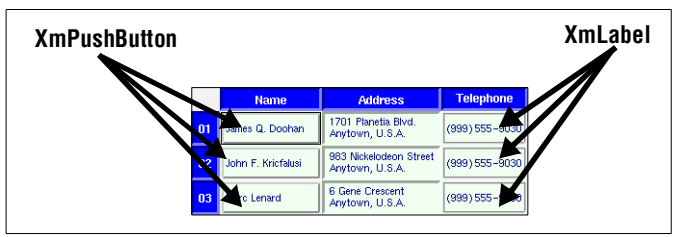


Figure 59 Setting widgets in cells

Widgets in cells/labels are useful for providing menus or buttons within a table. For example, the *entry* example (located in *$XRTHOME/src/table/examples*) uses

OptionMenus in one column to allow the user to select values from a specific list of choices.

6.1.1 Widgets in a Table

Adding a Widget

You can place a widget in a context of cells/labels by setting the `XmNxrtTblWidgetLocation` constraint resource on the widget or by setting the `XmNxrtTblWidgetContext` resource on the table.

The following example uses `XmNxrtTblWidgetLocation` to place a widget in a cell:

```
button = XtVaCreateWidget("button",
    xmArrowButtonWidgetClass,   table,
    XmNarrowDirection,          XmARROW_UP,
    XmNxrtTblWidgetLocation, XrtTblSetContext(1, 5),
    NULL);
```

And this sets `XmNxrtTblWidgetContext`:

```
title = XtVaCreateWidget("title",
    xtXrtLabelWidgetClass,      table,
    XmNlabelString,             "Employee Name",
    NULL);
XtVaSetValues(button,
    XmNxrtTblContext, XrtTblSetContext(XRTTBL_LABEL, 0),
    XmNxrtTblWidgetContext,     title,
    NULL);
```

Widgets should be created unmanaged because the table will control their management when they are set in the table.

To place a widget in the intersection of the row and column label areas, set the context to `(XRTTBL_LABEL, XRTTBL_LABEL)` before setting `XmNxrtTblWidgetContext`.

Widget Cloning

XRT/table creates identical copies (called *clones*) of widgets set in a cell or label in the following cases:

- When the current context is more than a single cell/label (for example, an entire column) when the widget is added (using `XmNxrtTblWidgetContext`).

Clones are created when their location is specified by setting `XmNxrtTblWidgetLocation` (or `XmNxrtTblWidgetContext`). This means that all clone resources, including callbacks, must be set before then. For example, here is code that creates enough clones to fill a column.

```
button = XtVaCreateWidget("button",
    xmArrowButtonWidgetClass,   table,
    XmNarrowDirection,          XmARROW_UP,
    NULL);
XtAddCallback(table, XmNxrtTblDisplayWidgetCallback,
    buttonCB, NULL);
XtVaSetValues(button,
    XmNxrtTblWidgetLocation,
                XrtTblSetContext(XRTTBL_ALLCELLS, 5),
    NULL);
```

In this example, the *buttonCB()* callback modifies the resources of the cloned arrow button.

The following widget classes can be cloned:

- Primitive widgets–XmArrowButton, XmDrawnButton, XmLabel, XmList, XmPushButton, XmScrollBar, XmSeparator, XmText, XmTextField, XmToggleButton, XmXrtField (and its subclasses), XtXrtLabel, XtXrtGraph, XtXrt3d
- Composite widgets–XmBulletinBoard, XmCommand, XmDrawingArea, XmFileSelectionBox, XmForm, XmFrame, XmMainWindow, XmMessageBox, XmPanedWindow, XmRowColumn (except the OptionMenu), XmScrolledWindow, XmSelectionBox, XmXrtComboBox, XmXrtSpinBox, XtXrtTable

The table dynamically creates as many clones as there are visible cells. The master widget – the widget being cloned – is never mapped to a cell. The cloned widgets are cycled as a user scrolls through the table. As the number of visible rows/columns changes, the table creates clones as necessary.

The `XmNxrtTblArgListContext` resource and DisplayWidget callback (specified by `XmNxrtTblDisplayWidgetCallback`) allow you to program cloned widgets individually. (Note that resources for cloned widgets cannot be set after the widget location is specified by `XmNxrtTblWidgetLocation` or `XmNxrtTblWidgetContext`, as the clones are created at the time the location is specified. This behavior differs from that of earlier versions of XRT/table.)

Note: Widgets cannot be cloned using `XrtTblRangeContext`.

Resources set on Widgets

Cloned widgets are identical to the widget they were cloned from except for the cell/label resources applied to them. XRT/table applies several of its resources to the corresponding resources of some widget classes when set in a cell or label. To set these widget resources, you must set the table resource on the cell that contains the widget. The following shows the table resources that are applied to widgets set in a cell/label:

Widgets	Table Resources: XmNxrtTblAlignmentContext	XmNxrtTblBackgroundContextt	XmNxrtTblCellValueContext	XmNxrtTblCellValueContext	XmNxrtTblEditableContext	XmNxrtTblFontListContext	XmNxrtTblForegroundContext	XmNxrtTblLabelContext	XmNxrtTblMaxLengthContext	XmNxrtTblMultilineContext	XmNxrtTblPixelHeightContext	XmNxrtTblPixelWidthContext	XmNxrtTblPixmapContext[a]	XmNxrtTblTraversableContext
XmLabel	✔	✔	✔			✔	✔	✔			✔	✔	✔	✔
XmPushButton	✔	✔	✔			✔	✔				✔	✔	✔	✔
XmText		✔		✔	✔	✔	✔		✔	✔	✔	✔	✔	✔
XmTextField		✔			✔	✔	✔		✔		✔	✔	✔	✔
XmXrtField (and subclasses)		✔		✔	✔	✔	✔		✔	✔	✔	✔	✔	✔
XtXrtLabel	✔	✔	✔			✔	✔	✔			✔	✔	✔	✔
XtXrtTable		✔					✔				✔	✔	✔	✔
Any other widget		✔					✔				✔	✔	✔	✔
	Widget Resources: XmNalignment	XmNbackground	XmNlabelString	XmNvalue	XmNeditable	XmNfontList	XmNforeground	XmNlabelString	XmNmaxLength	XmNeditMode	XmNheight	XmNwidth	XmNbackgroundPixmap	XmNtraversalOn

a. Pixmaps are applied in a tile pattern, regardless of the setting of `XmNxrtTblPixmapLayoutContext`.

When a table resource is set on a cell/label containing a widget, XRT/table sets the widget's corresponding resource to match.

Variable Rows/Columns

When a widget is set in a variable height row or variable width column, XRT/table will resize the row/column to fit the widget's preferred size (determined by the *XtQueryGeometry()* method).

Unmanaging a Widget

To unmanage a widget (remove a widget from the table temporarily), set the `XmNxrtTblWidgetContext` corresponding to the widget to `NULL` or set the widget's `XmNxrtTblWidgetLocation` to `NULL`.

When a widget is unmanaged it is not destroyed, but all clone widgets will be destroyed. Deleting the table cell containing a widget, for instance using *XrtTblDeleteRows()*, has the same result; the widget is unmapped and its clones are destroyed.

Note: Unlike cloning a widget, removing a widget is possible with `XrtTblRangeContext`.

Destroying a Widget

To destroy a widget that is a child of the table, call the *XtDestroyWidget()* method. If any cloned widget is destroyed, any clones that have been created are also destroyed.

If the destroyed widget is contained in the `XmNxrtTblWidgetSeries` list, that element is removed from the series.

Determining Widget Location

Use the *XrtTblGetRowColByWidget()* method when your application needs to determine the cell/label location containing a given widget. This is useful when the same callback routine is called by several cloned widgets.

Using an ArgList

You can use `XmNxrtTblArgListContext` to store widget resource settings on a cell-by-cell basis. This is a useful way to program attributes of cloned widgets individually.

Note: Do not free the ArgList after setting `XmNxrtTblArgListContext` because, unlike most of the XRT/table resources, it is not copied internally.

CreateWidget Callback

A CreateWidget callback (specified with `XmNxrtTblCreateWidgetCallback`) must be used to create or clone a widget when XRT/table does not know the widget's class pointer (such as Motif OptionMenus) or is unable to clone the widget. You can also use it to set attributes for each widget when it is created or cloned. Each callback is passed a pointer to the following structure:

```
typedef struct {
    XrtTblReason    reason;             // Read-only
    XEvent          *event;             // Read-only
    int             row, column;        // Read-only
    Widget          parent;             // Read-only
    WidgetClass     widget_class;       // Read-only
    String          class_name, name;   // Read-only
    Widget          clone_source;       // Read-only
    Widget          new_widget;
} XrtTblCreateWidgetCallbackStruct;
```

A CreateWidget callback can change the *new_widget* member when the *reason* member is `XRTTBL_REASON_CREATE_WIDGET_BEGIN`. To create a widget that

XRT/table does not recognize (*widget_class* is `NULL`), the callback would create the widget and set *new_widget* to it when *reason* is `XRTTBL_REASON_CREATE_WIDGET_BEGIN`.

The CreateWidget callback must be added when the table is created, because that is the point when resources specified in a resource file are applied to the table. *XtAddCallback()* adds the callback after the table is created. The following example shows how to add the callback when creating the table:

```
static XtCallbackRec createWidgetCBlist[] = {
    {(XtCallbackProc) createWidgetCB, NULL},
    {(XtCallbackProc) NULL, NULL}};
...
table = XtVaCreateManagedWidget("table",
    xtXrtTableWidgetClass,          parent,
    XmNxrtTblCreateWidgetCallback,  createWidgetCBlist,
    NULL);
```

DisplayWidget Callback

You can use a DisplayWidget callback (specified with `XmNxrtTblDisplayWidgetCallback`) to set attributes on a widget before it is displayed. DisplayWidget callbacks are called after applying ArgList values (specified by `XmNxrtTblArgListContext`) to it. Each callback is passed a pointer to the following structure:

```
typedef struct {
    XrtTblReason    reason;                              // Read-only
    XEvent          *event;                              // Read-only
    int             row, column;                         // Read-only
    Widget          clone_source;                        // Read-only
    Widget          widget;
} XrtTblDisplayWidgetCallbackStruct;
```

A DisplayWidget callback can set the *widget* member to a different widget.

If a widget is cloned, `XmNxrtTblDisplayWidgetCallback` cannot be set after the widget location is specified by `XmNxrtTblWidgetLocation` or `XmNxrtTblWidgetCallback`, as the widget is cloned as soon as the location is specified.

To use a DisplayWidget callback on widgets specified in resource files, it must be added to the table when it is created, as shown by the following:

```
static XtCallbackRec displayWidgetCBlist[] = {
    {(XtCallbackProc) displayWidgetCB, NULL},
    {NULL, NULL}};
...
table = XtVaCreateManagedWidget("table",
    xtXrtTableWidgetClass,          parent,
    XmNxrtTblDisplayWidgetCallback, displayWidgetCBlist,
    NULL);
```

WidgetSetValue Callback

You can use the `XmNxrtTblWidgetSetValueCallback` resource to create a list of callbacks to be called when a widget in a cell or label has resources set. This lets the application disable or modify the resource values.

Each callback in the list is passed the following structure:

```
typedef struct {
    int         reason;                    // Read-only
    XEvent      *event;                    // Read-only
    int         row, column;               // Read-only
    Widget      widget;                    // Read-only
    Arg         *arglist;                  // Read-only
    int         num_arglist;               // Read-only
    Boolean     doit;
} XrtTblWidgetSetValueCallbackStruct;
```

Each callback is called both before the resource values are set and after. *reason* specifies which condition is currently in effect, and is either `XRTTBL_REASON_SETVALUE_BEGIN` (before resources set) or `XRTTBL_REASON_SETVALUE_END` (after resources set).

The *doit* member allows you to override the resource-setting operation. If *doit* is set to `False`, the resources specified in *arglist* are not updated. By default, *doit* is `True` if the widget specified by *widget* is an OSF widget, and is `False` otherwise.

Shadows and Widgets

When a widget appears in a cell or label, you can specify that the cell/label shadow is to be drawn (if, for example, the widget is inset). To do this, set the `XmNxrtTblWidgetShadowContext` resource to `True`:

```
XtVaSetValues(table,
    XmNxrtTblContext,XrtTblSetContext(1,1),
    XmNxrtTblWidgetShadowContext, True);
```

By default, `XmNxrtTblWidgetShadowContext` is `False`, and the cell or label shadow is not drawn. (The `XmNxrtTblShadowThickness` resource controls the thickness of cell and label shadows.)

Examples

For a simple example of using widgets in cells, see the *entry* example, located in *$XRTHOME/src/table/examples*. A demo using more complex widgets can be found in *$XRTHOME/src/table/demos/widgets*.

6.1.2 Widgets in Resource Files

XRT/table will actually create widgets specified in resource files if it knows the widget's class pointer (specified by `XmNxrtTblWidgetClassList`). For example, the following resource file fragment creates and clones XmPushButton widgets for all of the cells in a column:

```
*.XtXrtTable.xrtTblWidgetSeries: \
        (allcells 4 XmPushButton.mybutton)
```

Widget Class List

`XmNxrtTblWidgetClassList` specifies the widget classes and class pointers of widgets that XRT/table can create from a resource file. This list is made up of the following widgets:

- Primitive widgets–XmArrowButton, XmDrawnButton, XmLabel, XmList, XmPushButton, XmScrollBar, XmSeparator, XmText, XmTextField, XmToggleButton, XmXrtField (and its subclasses), XtXrtLabel
- Composite widgets–XmBulletinBoard, XmCommand, XmDrawingArea, XmFileSelectionBox, XmForm, XmFrame, XmMainWindow, XmMessageBox, XmPanedWindow, XmRowColumn, XmScale, XmScrolledWindow, XmSelectionBox, XmXrtComboBox, XmXrtSpinBox, XtXrtTable

To specify a third-party widget in a resource file, add its class name and class pointer to `XmNxrtTblWidgetClassList` when you create the table.[1] The following code adds XRT/graph to the widget class list:

```
static XrtTblWidgetClass class_list[] = {
    {"XtXrtGraph", &xtXrtGraphWidgetClass},
    {NULL, NULL}};
...
table = XtVaCreateManagedWidget("table",
    xtXrtTableWidgetClass,          parent,
    XmNxrtTblWidgetClassList,       class_list,
    ...
    NULL);
```

6.2 Cell/Label Value Validation

Overview

XRT/table can be set to intercept and validate cell and label values before writing them to the table's cell and label values structures. Edited text is validated against a "data type" you specify for the cell/label. "Data types" are format and content rules for the text; they are not C language data types. Cell and label values are always represented internally as XmString.

The table uses its own ValidateCell callback to validate cell/label values. The callback calls the validation method corresponding to the cell's data type. The validation method parses the edited cell/label value, and sets the ValidateCell callback structure's *doit* member to `False` if the value does not conform to the data type.

XRT/table also allows you to use XRT/field widgets to edit and validate values. For details on how to do this, see section 6.3 on page 119.

Cell/label Data Type

Use `XmNxrtTblDataTypeContext` to set the data type of cells and labels. If set to one of the built-in types[2], XRT/table will automatically validate editing according to the validation policy (specified by `XmNxrtTblValidatePolicy`). The default data type,

1. `XmNxrtTblWidgetClassList` must be set when creating the widget because that is when resources specified in resource files are applied to the table.

2. You can also set `XmNxrtTblDataTypeContext` to any XRT/field widget type. This is described in section 6.3 on page 119.

String, causes no validation to be performed (anything is a valid String). The following lists the built-in data types and their validation methods:

Resource Value	Data Type	Validation Method
XRTTBL_TYPE_STRING	Any text string	No method called
XRTTBL_TYPE_DOUBLE	Double-precision value	*XrtTblCheckIsDouble()*
XRTTBL_TYPE_INT	Integer value	*XrtTblCheckIsInt()*
XRTTBL_TYPE_FLOAT	Floating-point value	*XrtTblCheckIsFloat()*
XRTTBL_TYPE_BOOLEAN	Any Boolean value	*XrtTblCheckIsBoolean()*

Validation Policy

The `XmNxrtTblValidatePolicy` resource determines the type of editing that triggers validation. By default, only cells edited by a user are validated. When set to `XRTTBL_VALIDATE_ALWAYS`, any change to cells or labels is validated, whether made by a user, or by the application. When set to `XRTTBL_VALIDATE_NEVER`, no validation is performed. When set to `XRTTBL_VALIDATE_SET`, only changes made by an application are validated.

Extending Data Types

You can extend the built-in data types by adding your own data types and validation methods to the list specified by `XmNxrtTblDataTypeList`. Once added to the list, you can set cells/labels to this data type (using `XmNxrtTblDataTypeContext`), and the validation method you define will be called to validate changes. The following example defines a positive integer data type and adds it to the data type list:

```
#define TYPE_INT_POSITIVE (XRTTBL_TYPE_LAST + 1)
static XrtTblDataTypeRoutine datatype_list[] = {
    {TYPE_INT_POSITIVE, (XtCallbackProc) checkIsPositive},
    {0,     NULL}
};
...
table = XtVaCreateManagedWidget("table",
    xtXrtTableWidgetClass,      toplevel,
    XmNxrtTblDataTypeList,      datatype_list,
    ...
    NULL);
```

Note that data types you add should be indexed from the `XRTTBL_TYPE_LAST` special value to avoid data types added in future releases of XRT/table conflicting with those you've defined.

The validation method you define should have the following prototype:

```
void
my_validation_method(
    Widget          table,
    XtPointer       call_data,
    XtPointer       cbs
    )
```

cbs is a pointer to a `XrtTblValidateCellCallbackStruct` structure (described in detail in the next section). This structure contains members for the edited value (*value*), a converted value (*converted_value*), whether the edited value is valid (*doit*), and whether to ring the terminal bell (*bell*)–the validation method can read and change some of these values to indicate whether the edited value is valid or not.

The following validation method checks that values are positive integers:

```
void
checkIsPositive(Widget table, XtPointer call_data,
        XtPointer client_data)
{
    XrtTblValidateCellCallbackStruct *cbs
      = (XrtTblValidateCellCallbackStruct *) client_data;

    /* Check whether a valid integer */
    XrtTblCheckIsInt(table, call_data, cbs);
    if (cbs->doit && cbs->converted_value < 0) {
        cbs->doit = False;
        cbs->bell = True;
    }
}
```

The source code for all validation methods can be found in *$XRTHOME/src/table/util*. You can use these to create your own methods.

ValidateCell Callback

The ValidateCell callback is primarily used by data validation methods. An application can also add a ValidateCell callback that is called after any validation methods using `XmNxrtTblValidateCellCallback`. This can be used to capture a failed data type validation and attempt to get more information from the user. Each callback is passed a pointer to the following structure:

```
typedef struct {
    XrtTblReason    reason;                          // Read-only
    XEvent          *event;                          // Read-only
    int             row, column;                     // Read-only
    Boolean         value_changed;                   // Read-only
    XrtTblDataType  datatype;                        // Read-only
    XtArgVal        *converted_value;                // Read-only
    XtPointer       user_data;                       // Read-only
    String          value;
    Boolean         doit;
    Boolean         bell;
} XrtTblValidateCellCallbackStruct;
```

Further Examples

The *spreadsheet* demo, located in *$XRTHOME/src/common_demos/spreadsheet*, contains examples of using the built-in data types, as well as defining and checking for new data types.

6.3 Using XRT/field with XRT/table

By default, an XmText widget is used to edit and validate text. XRT/table allows you to use XRT/field's validation, editing and display capabilities. To do this, perform the following four steps:

1. Ensure that the XRT/field library (`libxrtfield.a`) appears before the XRT/table library (`libxrttable.a`) in the list of libraries you specify when linking your compiled program.
2. Call the *XrtTblEnableXrtField()* procedure before creating any tables:

```
XrtTblEnableXrtField();
...
table = XtVaCreateManagedWidget("table",
    xtXrtTableWidgetClass, toplevel,
    NULL);
```

3. Use the `XmNxrtTblDataTypeContext` resource to specify the type of XRT/field widget you want to use. The following types are available:

`XRTTBL_TYPE_FIELD_CURRENCY`	currency field
`XRTTBL_TYPE_FIELD_DATE`	date/time field
`XRTTBL_TYPE_FIELD_FLOAT`	floating-point field
`XRTTBL_TYPE_FIELD_INT`	integer field
`XRTTBL_TYPE_FIELD_NONE`	
`XRTTBL_TYPE_FIELD_STRING`	string field
`XRTTBL_TYPE_FIELD_COMBO_CURRENCY`	currency combo box
`XRTTBL_TYPE_FIELD_COMBO_DATE`	date combo box
`XRTTBL_TYPE_FIELD_COMBO_FLOAT`	floating-point combo box
`XRTTBL_TYPE_FIELD_COMBO_INT`	integer combo box
`XRTTBL_TYPE_FIELD_COMBO_STRING`	string combo box
`XRTTBL_TYPE_FIELD_SPIN_CURRENCY`	currency spin box
`XRTTBL_TYPE_FIELD_SPIN_DATE`	date spin box
`XRTTBL_TYPE_FIELD_SPIN_FLOAT`	floating-point spin box
`XRTTBL_TYPE_FIELD_SPIN_INT`	integer spin box
`XRTTBL_TYPE_FIELD_SPIN_STRING`	string spin box

An XRT/field widget of the appropriate type is automatically created when needed. Only one of each type of field is created, and is reused as the user traverses.

4. To set resources for your XRT/field widget, use the `XmNxrtTblFieldArgListContext` resource. Its value must be a static list of Args, and the last element of the list must be `NULL`:

```
static Arg args[3];

XtSetArg(args[0], XmNxrtFldMask, "###-####");
XtSetArg(args[1], XmNxrtFldAllowNull, False);
XtSetArg(args[2], NULL, NULL);
XtVaSetValues(table,
    XmNxrtTblContext, XrtTblSetContext(5, 5),
    XmNxrtTblFieldArgListContext, args,
    NULL);
```

When editing is completed, the edited value is stored in the table's `XrtTblCellValues` structure.

The *table_entry* demo, located in *$XRTHOME/src/common_demos/table_entry*, provides an example of a program that uses XRT/field widgets with XRT/table.

Note: It is up to the user to manage the memory for the `XmNXrtTblFieldArgList`, as it is not copied internally.

Fields in Resource Files

The name of a created XRT/field widget reflects its type:

Widget Type	Field Name	Child Field Name	Child Arrow Name	Child Arrow Name
XRTTBL_TYPE_FIELD_CURRENCY	"currency_field"			
XRTTBL_TYPE_FIELD_DATE	"date_field"			
XRTTBL_TYPE_FIELD_FLOAT	"float_field"			
XRTTBL_TYPE_FIELD_INT	"int_field"			
XRTTBL_TYPE_FIELD_STRING	"string_field"			
XRTTBL_TYPE_FIELD_COMBO_CURRENCY	"combo_currency_field"	"field"	"arrow"	
XRTTBL_TYPE_FIELD_COMBO_DATE	"combo_date_field"	"field"	"arrow"	
XRTTBL_TYPE_FIELD_COMBO_FLOAT	"combo_float_field"	"field"	"arrow"	
XRTTBL_TYPE_FIELD_COMBO_INT	"combo_int_field"	"field"	"arrow"	
XRTTBL_TYPE_FIELD_COMBO_STRING	"combo_string_field"	"field"	"arrow"	
XRTTBL_TYPE_FIELD_SPIN_CURRENCY	"spin_currency_field"	"field"	"decr_arrow"	"incr_arrow"
XRTTBL_TYPE_FIELD_SPIN_DATE	"spin_date_field"	"field"	"decr_arrow"	"incr_arrow"
XRTTBL_TYPE_FIELD_SPIN_FLOAT	"spin_float_field"	"field"	"decr_arrow"	"incr_arrow"
XRTTBL_TYPE_FIELD_SPIN_INT	"spin_int_field"	"field"	"decr_arrow"	"incr_arrow"
XRTTBL_TYPE_FIELD_SPIN_STRING	"spin_string_field"	"field"	"decr_arrow"	"incr_arrow"

Resources for this field can be set in a resource file.

Retrieving a Field

To retrieve an XRT/field widget of a particular type, call the *XrtTblGetXrtFieldByType()* method:

```
field = XrtTblGetXrtFieldByType(table,
        XRTTBL_TYPE_FIELD_CURRENCY);
```

If a field of the appropriate type does not exist, it is created (unmanaged).

Once you have retrieved a field, you can use it to format or update a cell value. For example, the following code converts an integer value to hexadecimal format and puts it in a cell:

```
field = XrtTblGetXrtFieldByType(table,
        XRTTBL_TYPE_FIELD_INT);
XtVaSetValues(field,
    XmNxrtFldDisplayFormat, XRTFLD_FORMAT_HEX,
    XmNxrtFldInternalValue, 53,
    NULL);
value = XrtFldGetDisplayValue(field);
XtVaSetValues(table,
    XmNxrtTblContext,XrtTblSetContext(4, 3),
    XmNxrtTblCellValueContext,value,
    NULL);
XtFree(value);
```

The XRT/field method *XrtFldGetDisplayValue()* retrieves the string that the field would display. This string is then stored in the cell by setting the `XmNxrtTblCellValueContext` resource.

You cannot set the following XRT/field resources through a resource file or through a call to *XtSetValues()* on the field:

```
XmNxrtFldMask
XmNxrtFldPickList
XmNxrtFldMatchPickList
XmNxrtFldPickListIndex
```

To set these resources, you need to use `XmNxrtTblFieldArgListContext`, since these resources are reset internally by the table as the user moves from cell to cell in the table. To have them take affect in the new cell, they have to be reloaded from `XmNxrtFldArgListContext`, with the exception of `XmNxrtFldPickList`. The table uses its own picklist resources (such as `XmNxrtTblStringPickListContext`) and when a user moves into a new cell, the table's own picklist resources are applied to the field. If no XRT/table picklist is provided for the cell, then the picklist from the `XmNxrtFldArgListContext` is used, if present. If the picklist exists in both the table and the `XmNxrtTblFieldArgListContext`, the picklist in the table takes precedence.

It is strongly recommended that you let the table manage the picklists for you (using the various `XmNxrtTbl...PickList` resources), rather than using `XmNxrtTblFieldArgListContext` to specify picklists. If you decide to use the latter method, you will also have to allocate, maintain and deallocate the memory required for these picklists. XRT/table will not handle it for you.

Further XRT/field Information

For more information on XRT/field and its features, including value validation, refer to the *XRT/field Programmer's Guide and Reference Manual*.

6.4 Table Sizing and Resizing

The table can be created as a child of any of the Motif Composite widgets. Usually the parent Composite widget determines the appropriate size for the table widget, and the table widget adjusts accordingly.

The table determines the sizing of each of its components by considering the following factors:

- The window's current width and height.
- The size and position of each of the titles. This, in turn, depends on the size of the widget used for each title, its location in the table, and its offset.

The table's cells and labels fill the remaining rectangular area. If there is not enough space to display all of the rows or columns in the table, scrollbars appear, to enable a user to scroll all rows/columns in the table into view within this cell/label viewport.

Resizing Approaches

Changing the value of the resources that affect the row/column size (see Figure 60) can cause the table to attempt to resize, depending on the value of `XmNxrtTblAllowResize`. How you set this resource depends on whether you want to control the amount of the table displayed or the size of the widget.

When set to `XRTTBL_RESIZE_NONE`, the table will not attempt to resize when the resources listed in Figure 60 change.[1] The size of the cell/label viewport remains the same.

When set to `XRTTBL_RESIZE_ALL`, changes to the cell/label resources listed in Figure 60 cause the table to attempt to resize so that the cell/label viewport shows the same amount of the table.

When set to `XRTTBL_RESIZE_HORIZONTAL` or `XRTTBL_RESIZE_VERTICAL`, changes to the cell/label resources listed in Figure 60 cause table resize attempts in one direction only.

1. Except setting `XmNxrtTblVisibleColumns` and `XmNxrtTblVisibleRows`, which always cause resize attempts.

Setting the following resources can cause resize attempts:

XmNxrtTblCellValueContext	XmNxrtTblCharHeight[CS]	XmNxrtTblCharWidth[CS]
XmNxrtTblColumnLabels	XmNxrtTblFontList[CS]	XmNxrtTblLabelContext
XmNxrtTblMarginHeight	XmNxrtTblMarginWidth	XmNxrtTblPixelHeight[CS]
XmNxrtTblPixelWidth[CS]	XmNxrtTblPixmap[CS]	XmNxrtTblRowLabels
XmNxrtTblShadowThickness	XmNxrtTblWidget[CS]	

(Resources ending in [CS] indicate both the Context and Series version of the resource, for example XmNxrtTblFontListContext and XmNxrtTblFontListSeries.)

Setting XmNxrtTblVisibleColumns or XmNxrtTblVisibleRows always cause resize attempts.

Figure 60 Resources that can cause table resize attempts

Note: Whether the table is actually able to resize itself depends on whether its parent widgets allow the resize. The shell widget's XmNallowShellResize resource must also be True. When a table's resize request is refused by any parent widget, the table displays or updates its scrollbars instead.

Expose and Resize Callbacks

An application can find out when the table widget is exposed or resized by specifying a callback routine using the XmNxrtTblExposeCallback or XmNxrtTblResizeCallback resources. You can use Expose callbacks to draw onto the table image using Xlib procedures. You can use Resize callbacks to adjust the widget's resources, depending on the size of the table.

The following Resize callback resizes all columns to fit the resized table width:

```
static void
resizeCB(
    Widget          table,
    XtPointer       data,
    XrtTblResizeCallbackStruct *cbs)
{
    int ncols, colwidth, firstwidth;
    XrtTblSeries *series;

    if (cbs->reason != XRTTBL_REASON_RESIZE_END)
        return;
    XtVaGetValues(table,
        XmNxrtTblNumColumns,  &ncols, NULL);
    ncols++;                    /* Include row labels */
    colwidth = cbs->width/ncols;
    firstwidth = cbs->width - (ncols - 1) * colwidth;
```

```
    series = XrtTblSeriesCreate(XRTTBL_VALUE_INT);
    XrtTblSeriesAdd(series,
        XrtTblSetContext(XRTTBL_ALL, XRTTBL_ALL),
        colwidth);
    XrtTblSeriesAdd(series,
        XrtTblSetContext(XRTTBL_ALL, XRTTBL_LABEL),
        firstwidth);
    XtVaSetValues(table,
        XmNxrtTblDisplayVertScrollBar, XRTTBL_DISPSB_NEVER,
        XmNxrtTblPixelWidthSeries,    series,
        XmNxrtTblAllowResize,   XRTTBL_RESIZE_HORIZONTAL,
        NULL);
    XtVaSetValues(table,
        XmNxrtTblAllowResize,         XRTTBL_RESIZE_NONE,
        NULL);
    XrtTblSeriesDestroy(series);
}
```

To register the above callback routine, use the following call:

```
XtAddCallback(mytable, XmNxrtTblResizeCallback,
        resizeCB, NULL);
```

6.5 Compound Strings

XRT/table supports Motif Compound Strings. Compound Strings allow you to specify multi-font Strings, as shown in Figure 61. Note that you are not required to use Compound Strings when specifying cell values (unless `XmNxrtTblXmStringSetMode` is set; see section 3.5.1 on page 43 for details).

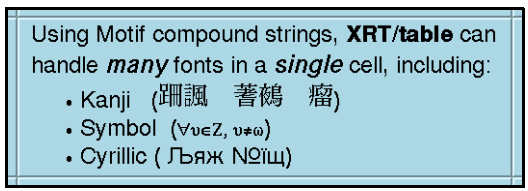


Figure 61 Using Compound Strings to display multiple fonts

Because the XmText widget cannot display Compound Strings, the current cell converts a Compound String to a String using the first font in the cell's fontlist. The cell is converted back to its original fonts when the user traverses out of the cell, unless the user edits the value.

Multi-font Compound Strings cannot be entered in a resource file. They must be created by an application program using *XmStringCreate()* with a tagged font list. See your Motif programming documentation for more information on programming Compound Strings.

The following example shows how to create a Compound String, and set it to a cell value:

```
static char *fallback[] = {
    "Simple*.fontList: -adobe-helvetica-medium-r-normal--
        25-180-100-100-p-130-iso8859-1=tag1, 9x15=b", NULL
};

int
main(int argc, char *argv[])

{
    Widget toplevel, table, label;
    XtAppContext app;
    XmString s1, s2, s3;

    XmFontListEntry ent1, ent2;
    XmFontList list;

    /* Initialize Intrinsics */
    toplevel = XtVaAppInitialize(&app, "Simple", NULL, 0, &argc, argv,
        fallback, NULL);

    ent1 = XmFontListEntryLoad(XtDisplay(toplevel),
        "-adobe-helvetica-medium-r-normal--25-180-100-100-p-130-iso8859-1",
        XmFONT_IS_FONT, "tag1");
    ent2 = XmFontListEntryLoad(XtDisplay(toplevel), "9x15",
        XmFONT_IS_FONT, "b");
    list = XmFontListAppendEntry(NULL, ent1);
    list = XmFontListAppendEntry(list, ent2);

    XrtTblSetXmStringConverter();

    s1 = XmStringCreate ("Italic", "tag1");
    s2 = XmStringCreate ("Regular", "b");
    s3 = XmStringConcat (s1,s2);

    /* Create the table */
    table = XtVaCreateManagedWidget("table",
        xtXrtTableWidgetClass,   toplevel,
        XmNxrtTblNumRows,        7,
        XmNxrtTblNumColumns,     5,
        XmNxrtTblCellValueContext,s3,
        XmNxrtTblDebug,          True,
        XmNxrtTblFontListContext,list,
        NULL);

    label = XtVaCreateManagedWidget ("label",
        xmXrtLabelWidgetClass,   table,
        XmNlabelString,          s3,
        XmNxrtTblLocation,       XRTTBL_LOCATION_BOTTOM,
        NULL);

    XmStringFree (s1);
    XmStringFree (s2);
    XmStringFree (s3);

    XtRealizeWidget(toplevel);
    XtAppMainLoop(app);
}
```

6.6 Series Resource Programming

When you set a Context resource, its corresponding Series resource is also updated. Series resources store all the Context resource settings over the life of the table. This section describes how to program the Series resources directly. Though more difficult than programming Context resources, there are situations when programming the Series resources is advantageous.

Components of a Series

The value of a Series resource is a stack of series members. All Series resources have a data type of `XrtTblSeries`, defined as follows:

```
typedef struct {
    XrtTblValueType     type;           /* value type */
    String              name;           /* resource name */
    int                 last_index; /* index of last item in list */
    int                 length;         /* # elements in list */
    XrtTblAnyValue      **list;         /* list */
} XrtTblSeries;
```

The `XrtTblValueType` and `XrtTblAnyValue` structures allow the structure to be used for any type of series (such as colors, Boolean values, and integer values). Each type of series is defined by its own structure. For example, the `XrtTblIntValue` structure is defined as follows:

```
typedef struct {
    XrtTblContext               context;
    int                         value;
} XrtTblIntValue;
```

Convenience Procedures

There are a number of procedures that manipulate `XrtTblSeries` structures. These procedures are fully documented in Appendix B on page 191.

XrtTblSeriesCreate()	Creates a new `XrtTblSeries` structure. A series must be created before members can be set or added to the series.
XrtTblSeriesCopy()	Creates a new `XrtTblSeries` and loads it with the members of an existing series.
XrtTblSeriesAdd()	Adds a member to a series given a context. Replaces existing context.
XrtTblSeriesAddValue()	Adds a member to a series at a position in the series list. Replaces value at existing position.
XrtTblSeriesSetValue()	Sets a series member at a given position.
XrtTblSeriesDelete()	Deletes a member from a series.
XrtTblSeriesFind()	Determines the position of the series member that contains the value being used for a given cell or label.
XrtTblSeriesIterate()	Iterates through the members of a series, returning the contents of each member.

XrtTblSeriesDestroy()	Destroys an `XrtTblSeries` structure and frees the memory allocated to it.

The following sections discuss cases when it is useful to use Series resources.

Attributes over several Contexts

When an application needs to change a resource over several parts of the table, it is useful to add all of the changes to one series and set the Series resource to that, as shown by the following example:

```
XtSetValues:

XrtTblSeries *series;
XtVaGetValues(table, XmNxrtTblPixelHeightSeries, &series, NULL);
series = XrtTblSeriesCopy(series);
XrtTblSeriesAdd(series, XrtTblSetContext(5, XRTTBL_ALL), 10);
XrtTblSeriesAdd(series, XrtTblSetContext(15, XRTTBL_ALL), 20);
XtVaSetValues(table, XmNxrtTblPixelHeightSeries, series, NULL);
XrtTblSeriesDestroy(series);
```

Fallback Resources

Series resources are useful for defining fallback resource values for different parts of a table in an application, as shown by the following:

```
static char *fallback[] = {
    "My_App*.XtXrtTable.background: #DBAFA7",
    "My_App*.XtXrtTable.xrtTblBackgroundSeries: \
            (ALL ALL Light Blue) (ALL LABEL #C29B93) \
            (LABEL ALL #C29B93)",
    "My_App*.XtXrtTable.xrtTblForegroundSeries: \
            (ALL ALL Black) (ALL LABEL White) (LABEL ALL White)",
    NULL,
};
toplevel = XtAppInitialize(&app_context, "My_App", NULL, 0,
            &argc, argv, fallback, NULL, 0);
```

Performance Considerations

When a resource has been set over many different parts of the table, it is often quicker to retrieve series values directly than to set a context and use *XtGetValues()*. The following example uses *XrtTblSeriesIterate()* to iterate through each member of the background color series, casting the returned value to the appropriate series type:

```
int row, column, position = 0;
XtArgVal value;
XrtTblSeries *bg_series;
Pixel bg_color, top_shadow;
XtVaGetValues(table, XmNxrtTblBackgroundSeries, &bg_series, NULL);
while (XrtTblSeriesIterate(bg_series, &position, &row, &column, &value)) {
        bg_color = (Pixel) value;
    ...
}
```

The following example uses *XrtTblSeriesFind()* to find the position in the series of a specified cell or label. The members can then be read directly.

```
int i;
char name[100];
if ((i = XrtTblSeriesFind(bg_series, 45, 34)) != XRTTBL_NOVALUE) {
        bg_color = (Pixel) bg_series->list[i]->color.value.pixel;
        top_shadow = (Pixel) bg_series->list[i]->color.value.top_shadow;
        strcpy(name, bg_series->list[i]->color.value.name);
}
```

6.7 Multiple Tables with Common Cell Values

Sometimes an application will want to create several table widgets that share the same `XrtTblCellValues` structure. This structure serves as a central data store for all the tables.

To accomplish this, one table displays the `XrtTblCellValues` structure; the others use a CellValue callback to display values from the first table's cell value structure.

If users should be able to edit the cells in any of the tables, you should use a ValidateCell callback to capture cell editing and update both the `XrtTblCellValues` structure and the values displayed by the other tables.

All of the above is shown by the following example:

```
static Widget table1, table2;

/* Cell value callback for table2 */
static void
cellvalueCB(Widget table, XtPointer data, XrtTblCellValueCallbackStruct *cbs)
{
    XmString value;
    XtVaSetValues(table1,
            XmNxrtTblContext, XrtTblSetContext(cbs->row, cbs->column),
            NULL);
    XtVaGetValues(table1, XmNxrtTblCellValueContext, &value, NULL);
    cbs->string = XmStringCopy(value);
}

/* Capture cell edit and set other table's value */
static void
validateCB(Widget table, XtPointer data, XrtTblValidateCellCallbackStruct *cbs)
{
    static Boolean entered = False;
    Widget other_table = (Widget) data;

    if (!cbs->value_changed)
            return;

    /* Prevent recursion after committing cell edit */
    if (entered)
            return;
    entered = True;

    /* Must commit cell edit, so cellvalueCB will get table1's cell value */
    if (table == table1)
            XrtTblCommitEdit(table, False);
    XtVaSetValues(other_table,
            XmNxrtTblContext, XrtTblSetContext(cbs->row, cbs->column),
            XmNxrtTblCellValueContext,          cbs->value,
            NULL);
    entered = False;
}
...
    table1 = XtVaCreateManagedWidget("table1",
            xtXrtTableWidgetClass,      parent,
            XmNxrtTblCellValues,        values,
            NULL);
    table2 = XtVaCreateManagedWidget("table2",
            xtXrtTableWidgetClass,      parent,
                            NULL);
    /* Register ValidateCell CB, using other table as call_data */
    XtAddCallback(table1, XmNxrtTblValidateCellCallback,
                            validateCB, (XtPointer) table2);
    XtAddCallback(table2, XmNxrtTblValidateCellCallback,
                            validateCB, (XtPointer) table1);
    /* Register CellValue CB for second table */
    XtAddCallback(table2, XmNxrtTblCellValueCallback,
                            cellvalueCB, (XtPointer) table2);
...
```

6.8 XRT/graph and XRT/3d Data

Developers using XRT/graph or XRT/3d can use these widgets with XRT/table. The *XrtTblGetGraphData()* procedure converts an XRT/graph array dataset or an XRT/graph data object handle to an `XrtTblCellValues` structure. The *XrtTblGet3dData()* procedure converts an XRT/3d dataset (either regular or irregular grid) to an `XrtTblCellValues` structure. Note that this does not apply to the scatter plot data type.

The following example converts an XRT/graph dataset for use in XRT/table:

```
XrtData *graph_data = XrtMakeDataFromFile(my_file);
XrtTblCellValues *cells = XrtTblGetGraphData(graph_data);
graph = XtVaCreateManagedWidget("graph",
        xtXrtGraphWidgetClass,          parent,
        XtNxrtData,                     graph_data,
        NULL);
table = XtVaCreateWidget("table",
        xtXrtTableWidgetClass,          parent,
        XmNxrtTblCellValues,            cells,
        NULL);

/* table copies values, so destroy them.
* Don't destroy graph data, since it doesn't copy them */
XrtTblDestroyCellValues(cells);
```

XRT/table includes two demo applications that use XRT/graph and XRT/3d, complete with source code. The *graph_editor* demo is located in *$XRTHOME/src/common_demos/graph_editor*. The *surface_editor* demo is located in *$XRTHOME/src/common_demos/surface_editor*.

6.9 Programming Drag and Drop

XRT/table allows you to change its drag and drop functionality and extend its capabilities programmatically. This section describes programming the following:

- Disallowing dragging from or dropping to widgets other than tables.
- Specifying the delimiter to use when dragging normal text into a table, or the delimiter to use when exporting normal text from a table.
- Using Export or Import callbacks.
- Defining your own drag and drop targets and their associated conversion procedures.

6.9.1 Drag and Drop Targets

The `XmNxrtTblExportTargets` and `XmNxrtTblImportTargets` resources allow you to control the types of drag and drop operations permitted in your tables.

`XmNxrtTblExportTargets` allows you to regulate the behavior of a table when it is being used as a drag source. The value of this resource is an array of pointers to data

items, each of which specifies one type of drag operation. Each data item in this list is called a *target*.

Similarly, XmNxrtTblImportTargets defines a list of targets which control the behavior of a table when it is being used as a drop site.

Both of these resources are passed an array of pointers to XrtTblImportExportStruct structures:

```
typedef struct {
    String  target;
    XrtTblImportExportConvertProc converter;
} XrtTblImportExportStruct;
```

Here, *target* is the name of the target type being defined by this structure member. *converter* specifies a conversion procedure to be used when performing this type of drag and drop operation. Conversion procedures are only needed when you are defining your own drag and drop targets, or when you wish to override the existing behavior. (Conversion procedures are discussed in section 6.9.4 on page 134. Application-defined drag and drop targets are discussed in section 6.9.4 on page 134.)

Targets can be divided into two types: targets which indicate a data format, and informational targets.

Data Format Targets

Data format targets indicate that text is being dragged from/dropped into either an XRT/table widget or an XmText/XmTextField widget. They are the only targets for which conversion procedures are defined. Each drag and drop operation defines one of the following targets:

XRTTBL_TGT_CELLS	Operation uses XRT/table data transfer format
XRTTBL_TGT_COMPOUND_TEXT XRTTBL_TGT_TEXT XRTTBL_TGT_STRING	Operation is to/from an XmText or XmTextField widget (or any other widget that understands these targets)

By default, all of these target types are included in both the XmNxrtTblImportTargets and XmNxrtTblExportTargets lists.

Informational Targets

Informational targets provide information about the data being dragged and dropped. The following targets are defined:

XRTTBL_TGT_RANGE	Data is a range of cells
XRTTBL_TGT_ROWS	Data is one or more entire rows (optionally including labels).

XRTTBL_TGT_COLUMNS	Data is one or more entire columns (optionally including labels)
XRTTBL_TGT_COLUMNLABELS	Column labels included with data
XRTTBL_TGT_ROWLABELS	Row labels included with data
XRTTBL_TGT_INTERNALMOVE	The drag and drop operation is from one part of a table to another part of the same table

When data is exported from a table, one of XRTTBL_TGT_RANGE, XRTTBL_TGT_ROWS or XRTTBL_TGT_COLUMNS is automatically added to the list specified by XmNxrtTblExportTargets, depending on the user's selection:

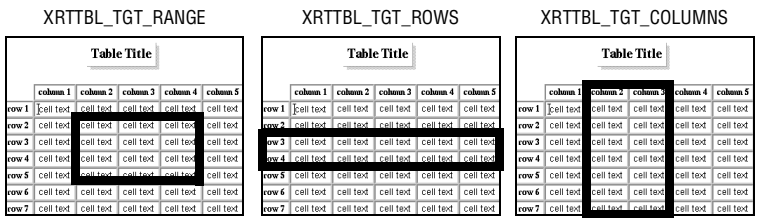


Figure 62 Drag and drop targets indicating cell/label selection

By default, XRTTBL_TGT_RANGE, XRTTBL_TGT_ROWS and XRTTBL_TGT_COLUMNS are included in both the XmNxrtTblImportTargets and the XmNxrtTblExportTargets lists. XRTTBL_TGT_INTERNALMOVE is automatically added to the XmNxrtTblExportTargets list if the drag and drop operation is contained within a table, and is included by default in the XmNxrtTblImportTargets list.[1]

Target List Procedures

The following XRT/table procedures manipulate drag and drop target lists:

- *XrtTblImportExportListCopy()* creates a copy of a target list.
- *XrtTblImportExportListAdd()* adds a target to the list.
- *XrtTblImportExportListDestroy()* destroys a target list.
- *XrtTblImportExportListDelete()* deletes a target from a target list.

6.9.2 Specifying the Text Delimiter

When normal text is imported into a table, the XmNxrtTblImportDelimiter resource defines the character string used to separate one cell's text from another. By default, this character string is "\t" (a single tab character):

1. Note that drag and drop operations within a table may not work properly if XRTTBL_TGT_INTERNALMOVE is removed from the target list specified by XmNxrtTblImportTargets.

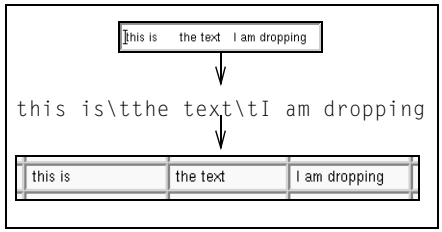


Figure 63 Delimited text dropped into cells

Setting this resource to XRTTBL_DELIMIT_BY_WHITESPACE indicates that any combination of spaces and tabs (of any length) is treated as a delimiter. To specify that no delimiter is to be written, set this resource to XRTTBL_DELIMIT_NONE.

The XmNxrtTblExportDelimiter resource defines the character string to be used as a delimiter when normal text is exported from a table. Its default is also "\t"; to specify no delimiter, set this resource to XRTTBL_DELIMIT_NONE.

To restore the default value for either delimiter resource, set it to XRTTBL_DELIMIT_DEFAULT or NULL.

6.9.3 Export and Import Callbacks

Export Callbacks

You can define callbacks to be invoked when dragging from a table. To do this, add to the callback list defined by the XmNxrtTblExportCallback resource. Each callback in this list is passed the following structure:

```
typedef struct {
    XrtTblReason        reason;
    XEvent              *event;
    int                 row, column;
    String              target;
    Boolean             internal;
    XrtTblRange         range;
    unsigned char       operation;
    Boolean             doit;
    Boolean             bell;
} XrtTblExportCallbackStruct;
```

These callbacks are invoked at the following times:

- When a mouse button is depressed to start a drag and drop operation (*reason* is XRTTBL_REASON_EXPORT_DRAG_BEGIN)
- When the drop is about to begin (*reason* is XRTTBL_REASON_EXPORT_TARGET)
- When the drop has completed (*reason* is XRTTBL_REASON_EXPORT_DRAG_END)

These callbacks allow you to do the following:

- Change the drag and drop operation from a copy to a move by changing the *operation* member (whose value is either XmDROP_MOVE or XmDROP_COPY)

- Suppress the operation by setting the *doit* member to `False` (if *reason* is `XRTTBL_REASON_EXPORT_DRAG_BEGIN` or `XRTTBL_REASON_EXPORT_TARGET`)
- Ring the terminal bell by setting *bell* to `True`

Import Callbacks

You can also define callbacks to be invoked when dropping into a table. To do this, add to the callback list defined by the `XmNxrtTblImportCallback` resource. Each callback is passed the following structure:

```
typedef struct {
    int             reason;                          // Read-only
    XEvent          *event;                          // Read-only
    int             row, column;                     // Read-only
    String          target;                          // Read-only
    Boolean         internal;                        // Read-only
    unsigned char   operation;
    unsigned char   dropSiteStatus;
    Boolean         insert;
    Boolean         doit;
    Boolean         bell;
} XrtTblImportCallbackStruct;
```

Import callbacks are invoked at the following times:

- When a drag is occurring (*reason* is `XRTTBL_REASON_IMPORT_DRAG_MOTION`)
- When the mouse button is released (*reason* is `XRTTBL_REASON_IMPORT_BEGIN`)
- When a drop conversion is about to start (*reason* is `XRTTBL_REASON_IMPORT_TARGET`)

operation, *doit*, *bell*, *row*, *column* and *internal* are identical to their equivalents in the `XrtTblExportCallbackStruct` structure. *dropSiteStatus* can be set to either `XmVALID_DROP_SITE` (drop valid at this site, which is the default) or `XmINVALID_DROP_SITE` (drop invalid). *insert* is used when normal text (text from an outside widget, or text from a single cell) is dropped into a cell; it can be set to `True` (insert if possible; the default) or `False` (overwrite existing data).

6.9.4 Defining Your Own Targets

XRT/table enables you to define your own drag and drop targets.[1] To do this, perform the following steps:

1. Pick a unique target name (e.g., "`MY_TARGET`").
2. Define a conversion method to read data in your target format.
3. Use your target name and conversion method to create an element of type `XrtTblImportExportStruct`:

```
XrtTblImportExportStruct mytarget =
    {"MYTARGET", MyTargetProc};
```

1. Before attempting to change drag and drop functionality, be sure that you are familiar with the contents of Chapter 18 of the *Motif Programming Manual* (volume 6A of the X Window System guides).

4. Use *XrtTblImportExportListAdd()* to add your new target to either the import targets list or the export targets list.
5. Set `XmNxrtTblImportTargets` or `XmNxrtTblExportTargets` to the new target list.

An export conversion method you create must expect to be passed the following arguments:

```
Boolean
ExportConversionProcedure(
    Widget          table,
    String          target,     /* Name of target type */
    XrtTblRange     *range,     /* Region of operation */
    String          *processed, /* Targets already processed */
    XtPointer       *data,      /* Returned, converted data */
    unsigned long   *data_len   /* Returned, length of data */
)
```

table is the table from which data is being exported. *target* is one of the targets listed above (or your own export target). *range* is the range of cells/labels from which data is being exported. *processed* is the list of targets already processed. *data* is the returned (converted) data exported; *data_len* is its length. *data* must be allocated using *XtMalloc()*, and will be freed automatically.

If you are creating an import conversion method, it must expect the following arguments:

```
Boolean
ImportConversionProcedure(
    Widget          table,
    String          target,     /* Name of target type */
    XrtTblContext   *context,   /* Region of operation */
    String          *processed, /* Targets already processed */
    Boolean         insert,     /* Insert data if possible */
    XtPointer       data,       /* Pointer to data to import */
    unsigned long   data_len    /* Length of data to import */
)
```

table is the table from which data is being exported. *target* is one of the targets listed above (or your own import target). *context* is the table context to which data is being imported. *processed* is the list of targets already processed. *insert* is used when normal text (text from an outside widget, or text from a single cell) is dropped into a cell; it can be set to `True` (insert if possible; the default) or `False` (overwrite existing data). *data* is the data to import; *data_len* is its length.

A conversion method does not need to be specified for the built-in targets. A default converter will be provided when necessary.

6.9.5 Data Transfer Format

The following is the format that XRT/table uses to transfer data when performing a drag and drop operation to another table widget:

```
DIM (rows)      (columns)
CLB (collabel) (collabel) ...
RLB (rowlabel)  (rowlabel) ...
XMS (celldata) (celldata) ...
...
```

The format for each line of this file is a three-letter keyword followed by one or more values enclosed in parentheses. Each item (keyword or value) on a line is separated from the next by a single blank space.

The first line, headed by `DIM`, lists the number of rows and columns being transferred. If column labels are being dragged, `CLB` lists them; the number of *collabel* values specified with `CLB` is equivalent to the number of *columns* specified by `DIM`. Similarly, `RLB` lists the row labels, and the number of *rowlabel* values is equivalent to the number of *rows*.

`XMS` lists the cell data itself. One `XMS` line is provided for each row transferred. The number of *celldata* entries in each line is equivalent to the number of columns being transferred.

For example, here is a simple table and its equivalent data transfer format:

	column 1	column 2	column 3
row 1	cell 1	cell 2	cell 3
row 2	cell 4	cell 5	cell 6

```
DIM (2) (3)
CLB (column 1) (column 2) (column 3)
RLB (row 1) (row 2)
XMS (cell 1) (cell 2) (cell 3)
XMS (cell 4) (cell 5) (cell 6)
```

Figure 64 A table and its data transfer format

XRT/table also allows you to use this format to define converters for existing table targets. To do this, delete the existing target definition from the target list you are using, and then add a new one which specifies your converter:

```
XrtTblImportExportStruct mytextconv =
    {XRTTBL_TGT_COMPOUND_TEXT, myTextRoutine};

XrtTblImportExportListDelete(implist, 0,
    XRTTBL_TGT_COMPOUND_TEXT);
XrtTblImportExportListAdd(implist, 0, &mytextconv);
XtVaSetValues(droptable,
    XmNxrtTblImportTargets, implist,
    NULL);
```

Normally, you will not need to define converters for existing XRT/table target types.

6.10 XRT/table Internationalization

ISO Latin1 Character Set

Any X Window font supported on your X Server may be used for the cell and label text. ISO Latin1 is the ISO standard character set for North American and West European languages. Since ISO Latin1 is a superset of ASCII, specifying a ISO Latin1 font will not change the behavior of ASCII applications. X Window fonts with names ending in *iso8859-1* are ISO Latin1 fonts.

2-byte Fonts

2-byte fonts such as Kanji may be displayed in cells and labels. 2-byte fonts are only supported in XmStrings.

Printing

XrtTblVaDrawPS() and *XrtTblDrawPS()* generate PostScript that uses the full ISO Latin1 character set if it is supported on the PostScript imaging device. With the exception of Japanese characters, the printing methods cannot print text that uses 2-byte fonts.

6.11 Performance Improvement

When a table contains a large amount of data, or its attributes change frequently over its life, performance of the table can degrade. This section describes how to improve table performance.

The *stocks* demo, located in the *$XRTHOME/src/table/demos/stocks* directory, demonstrates many of these methods for many XRT/table tasks (including adding rows, columns and values).

6.11.1 General Performance

- Almost all XRT/table Context resources have an equivalent Series resource. In many cases, setting this Series resource directly improves performance. An example of a program that builds a series resource can be found in *$XRTHOME/src/table/demos/form/view.c*.
- When defining contexts for setting resources, be sure to make them as large as possible, and also try to minimize the number of contexts you define. The best approach is to define a default context of `(ALL, ALL)` and then set individual contexts afterwards.

  For example, if the background color for most of your table cells is blue, with a few that are red or green, set the default background context to blue first:

```
XtVaSetValues(table,
    XmNxrtTblContext,
         XrtTblSetContext(XRTTBL_ALL, XRTTBL_ALL),
    XmNxrtTblBackgroundContext, "blue",
    NULL);
```

  Then, define individual contexts for your red and green cells.

- Whenever possible, use a single *XtVaSetValues()* call to set all applicable context resources.
- Reducing XmNxrtTblShadowThickness improves redraw speed.
- If your table sets a large number of context resources (that is, there are many different cells/labels with customized resources), you can improve performance by managing your own context lists instead of using series resources. To do this, set the XmNxrtTblContextValueCallback resource as described in section 6.11.5 on page 139.

6.11.2 Interactive Performance

- If a large table has few traversable cells, traversal performance can be improved by setting XmNxrtTblTraversableContext to False and customizing traversal using a XmNxrtTblTraverseCellCallback.
- You can greatly reduce the amount of network traffic XRT/table generates by setting XmNxrtTblTrackCursor to False. This turns off tracking of the mouse pointer. The mouse pointer will not change shape as the user moves it over the table.
- You can improve interactive performance when scrolling through a table with a large number of cloned widgets by using XRT/table's preview scrolling. When there are a lot of cloned widgets, scrolling through a table decreases performance since the XRT/table needs to create and position so much at once. Preview scrolling obviates the need to perform screen updates until the scroll is complete.

6.11.3 Display Performance

- By default, XRT/table redraws and recomputes a table whenever one of its resources is set. If your table does not need to be redrawn and recomputed when a particular resource is changed, set XmNxrtTblRepaint to XRTTBL_REPAINT_NOCALC. (For example: to add several new rows to your table, set XmNxrtTblRepaint to XRTTBL_REPAINT_NOCALC, call *XrtTblAddRows()* for each row, then reset XmNxrtTblRepaint to XRTTBL_REPAINT_ON.)
- To increase performance when using variable height rows or variable width columns, set XmNxrtTblPixelHeightContext/XmNxrtTblPixelWidthContext to a value other than XRTTBL_VARIABLE, make the changes you require to the cell values, then reset XmNxrtTblPixelHeightContext/XmNxrtTblPixelWidthContext to XRTTBL_VARIABLE.

Double Buffering

Double buffering is a graphics technique that reduces the amount of flashing perceived by a user when a table changes or scrolls. Display updates are first drawn to an off-screen area, then copied to the screen.

When XmNxrtTblDoubleBuffer is set to XRTTBL_DOUBLEBUFFER_CELL (the default), the table is double-buffered on a cell/label basis. When set to XRTTBL_DOUBLEBUFFER_WINDOW, the entire window is double-buffered.

Setting XmNxrtTblDoubleBuffer to XRTTBL_DOUBLEBUFFER_NONE disables it. While this can improve performance, the user may see the table "flash" as its attributes change or it is scrolled.

6.11.4 Cell Values Handling

- Using *XtVaSetValues()* in conjunction with the XmNxrtTblContext resource can hamper XRT/table performance, as *XtVaSetValues()* is actually called once for each cell in the specified context. A faster approach is to directly build the table's XrtTblCellValues structure, which stores all the XmStrings in the table. (The *XrtTblReadAscii()* method provides one way of doing this, as shown in the *simple.c* example described in section 1.3 on page 13.)
- You can use your own callbacks instead of the table's XrtTblCellValues structure to set cell values. For more information on the use of XmNxrtTblCellValueCallback, see section 4.5 on page 82.
- When XmNxrtTblXmStringSetMode is set to XRTTBL_SET_STRING, cell values in the XrtTblCellValues structure are stored internally as Strings. This dramatically reduces calculation and drawing times.
- Set XmNxrtTblPixelHeightContext/XmNxrtTblPixelWidthContext to an explicit number of pixels; or
- Set XmNxrtTblCharHeightContext/XmNxrtTblCharWidthContext to an explicit number of lines/characters.
- To update a single cell as fast as possible, use *XrtTblSetCell()*. Note that his routine does not resize the cell to fit the new data, even if XmNxrtTblPixelWidthContext is XRTTBL_VARIABLE.

6.11.5 Managing Contexts Programmatically

XRT/table lets applications keep track of their own contexts in place of using Context or Series resources. This improves performance if a table contains a large number of customized cells, such as a table with individual cell background colors.

To control context values at runtime, define a callback and add it to the list specified by the XmNxrtTblContextValueCallback resource. Each callback in this list is called every time a cell or label context value is needed (during table size calculations or drawing).

Each XmNxrtTblContextValueCallback routine is passed the following structure:

```
typedef struct {
    int             reason;                              // Read-only
    int             row, column;                         // Read-only
    XEvent          *event;                              // Read-only
    XrtTblAction    action;                              // Read-only
    XrtTblContextReason context_reason;                  // Read-only
    XrtTblContextValue value;
} XrtTblContextValueCallbackStruct;
```

The *action* member is either `XRTTBL_ACTION_CALCULATE` (callback called during size calculation and table initialization) or `XRTTBL_ACTION_DRAW` (callback called during table drawing).

The *row* and *column* members indicate the cell or label currently being examined.

context_reason indicates which context value is currently needed for the cell. Each context reason corresponds to an equivalent XRT/table context resource. *value* is the current value of this particular context reason, and is of type `XrtTblContextValue`. Callbacks change the context value for the cell by changing *value*. (*value* is not freed after use, and must point to static storage.)

`XrtTblContextValue` is defined as a union. Each possible context reason corresponds to one member of the union. The type of this union member is the same as the type of the value of the context reason's equivalent context resource.

The following table lists the possible values of *context_reason*, the equivalent XRT/table resource for each context reason, and the name of the member of the `XrtTblContextValue` union associated with the context reason:

Context Reason	Equivalent XRT/table Resource	Union Member Name
XRTTBL_CONTEXT_ALIGNMENT	XmNxrtTblAlignmentContext	alignment
XRTTBL_CONTEXT_ARGLIST	XmNxrtTblArgListContext	arglist
XRTTBL_CONTEXT_BACKGROUND	XmNxrtTblBackgroundPixelContext	background
XRTTBL_CONTEXT_BORDERSIDES	XmNxrtTblBorderSidesContext	border_sides
XRTTBL_CONTEXT_BORDERTYPE	XmNxrtTblBorderTypeContext	border_type
XRTTBL_CONTEXT_BOTTOMSHADOW	XmNxrtTblBottomShadowContext	bottom_shadow
XRTTBL_CONTEXT_CASE	XmNxrtTblCaseContext	string_case
XRTTBL_CONTEXT_CHAR_HEIGHT	XmNxrtTblCharHeightContext	char_height
XRTTBL_CONTEXT_CHAR_WIDTH	XmNxrtTblCharWidthContext	char_width
XRTTBL_CONTEXT_CLIP_PIXMAP	XmNxrtTblClipPixmapContext	clip_pixmap
XRTTBL_CONTEXT_CURRENCYPICKLIST	XmNxrtTblCurrencyPickListContext	picklist
XRTTBL_CONTEXT_DATATYPE	XmNxrtTblDataTypeContext	datatype
XRTTBL_CONTEXT_DATEPICKLIST	XmNxrtTblDatePickListContext	picklist
XRTTBL_CONTEXT_EDITABLE	XmNxrtTblEditableContext	editable
XRTTBL_CONTEXT_FIELD_ARGLIST	XmNxrtTblFieldArgListContext	field_arglist
XRTTBL_CONTEXT_FLOATPICKLIST	XmNxrtTblFloatPickListContext	picklist
XRTTBL_CONTEXT_FOREGROUND	XmNxrtTblForegroundPixelContext	foreground
XRTTBL_CONTEXT_INTPICKLIST	XmNxrtTblIntPickListContext	picklist

Context Reason	Equivalent XRT/table Resource	Union Member Name
XRTTBL_CONTEXT_MAXLENGTH	XmNxrtTblMaxLengthContext	maxlength
XRTTBL_CONTEXT_MULTILINE	XmNxrtTblMultilineContext	multiline
XRTTBL_CONTEXT_PIXEL_HEIGHT	XmNxrtTblPixelHeightContext	pixel_height
XRTTBL_CONTEXT_PIXEL_WIDTH	XmNxrtTblPixelWidthContext	pixel_width
XRTTBL_CONTEXT_PIXMAP	XmNxrtTblPixmapContext	pixmap
XRTTBL_CONTEXT_PIXMAP_LAYOUT	XmNxrtTblPixmapLayoutContext	pixmap_layout
XRTTBL_CONTEXT_PIXMAP_SHADOW	XmNxrtTblPixmapShadowContext	pixmap_shadow
XRTTBL_CONTEXT_PS_FONTLIST	XmNxrtTblPSFontListContext	ps_fontlist
XRTTBL_CONTEXT_SENSITIVE	XmNxrtTblSensitiveContext	sensitive
XRTTBL_CONTEXT_STRINGPICKLIST	XmNxrtTblStringPickListContext	picklist
XRTTBL_CONTEXT_TOP_SHADOW	XmNxrtTblTopShadowContext	top_shadow
XRTTBL_CONTEXT_USERDATA	XmNxrtTblUserDataContext	user_data
XRTTBL_CONTEXT_WIDGET_SHADOW	XmNxrtTblWidgetShadowContext	widget_shadow
XRTTBL_CONTEXT_WORDWRAP	XmNxrtTblWordWrapContext	wordwrap

For example, the following ContextValue callback sets the height of the third row of the table to 10 characters:

```
void
ContextValueCB(Widget table, XtPointer data,
    XrtTblContextValueCallbackStruct *cs);
{
    if (cs->context_reason == XRTTBL_CONTEXT_CHAR_HEIGHT &&
        cs->row == 2) {
        cs->value.char_height = 10;
    }
}
```

The type of the *char_height* member is the same as the data type of the `XmNxrtTblCharHeightContext` resource (that is, `int`).

Some context values cannot be set from within `XmNxrtTblContextValueCallback` routines. The following context resources cannot be set by a ContextValue callback (they have no equivalent context reason):

```
XmNxrtTblCellValueContext
XmNxrtTblFontListContext
XmNxrtTblOverflowContext
XmNxrtTblTraversableContext
```

Cell values can be set using `XmNxrtTblCellValueCallback`, which is described in section 4.5 on page 82. The other context values shown above must be set using context or series resources.

6.12 Warning and Error Message Customization

XRT/table uses *XtAppWarningMsg()* to output warning and error messages. The Xt error database allows you to customize the text of errors and warnings independent of the application. XRT/table's messages are provided in *$XRTHOME/include/Xm/XrtTable.err*.

To modify the warning and error messages, modify a copy of this file and either:

- Add the modified messages to the system message file in */usr/lib/X11/XtErrorDB*; or
- Merge the modified file with the system file at runtime by calling the Xlib procedure *XrmMergeDatabases()*.

6.13 Using C++ with XRT/table

XRT/table provides a C++ interface to the widget. To use it, include the header file *$XRTHOME/include/Xm/XrtTable.hxx* in your C++ program. This file defines an XrtTable class with the following methods:

- A constructor that calls *XmCreateXrtTable()*.
- A destructor that calls *XtDestroyWidget()*.
- A method for each of the C procedures that takes an XRT/table widget as its first argument. For example, the method for *XrtTblAddRows()* is *addRows()*.
- A method to set each settable resource. For example, the method to set `XmNxrtTblBackgroundContext` is *setBackgroundContext()*.
- A method to retrieve each resource. For example, the method to retrieve `XmNxrtTblBackgroundContext` is *getBackgroundContext()*.
- Methods to perform common Xt functions: *addCallback()*, *destroy()*, *getDisplay()*, *getName()*, *getParent()*, *isManaged()*, *isRealized()*, *manageChild()*.

A simple C++ example is found in *$XRTHOME/src/examples/simple_cpp.cxx*. The following example shows the use of C++ with XRT/table:

```
XrtTable *table = new XrtTable(toplevel, "table", NULL, 0);
table->setNumColumns(5);
table->setBackgroundContext("red");
table->addCallback(XmNxrtTblEnterCellCallback, enterCB, (XtPointer) table);
table->setContext(3, 4);
printf("name: %s,  color: %s\n",
          table->getName(), table->getBackgroundContext());
```

6.14 Using UIL with XRT/table

XRT/table allows you to use OSF's User Interface Language (UIL) to create table widgets. Since UIL only supports data types used by standard Motif widgets, some

XRT/table resources cannot be set using UIL, but can be set in an application with an `MrmNcreateCallback`. See Appendix H on page 265 for a complete list of UIL arguments and their data types.

Note that any Context resources set in a UIL file are applied to the entire table. To create a table with UIL:

- Include the XRT/table UIL header file *XrtTable.uil* in your application;
- Call *XrtTblMrmInitialize()* immediately after calling *MrmInitialize()*.

$XRTHOME/src/table/examples contains a short UIL example program–*styles_uil.c* and *styles_uil.uil*.

6.15 Reading and Writing Resource Files

XRT/table provides methods which allow to you save and read a table's resources and their current values.

Writing Resource Values

The *XrtTblWriteResources()* method appends the resources for a table to a file, as shown by the following example:

```
Widget  table;
FILE    *fp;

if (!(fp = fopen("mytable.res", "w"))) {
    perror("mytable.res");
    exit(0);
}
XrtTblWriteResources(table, fp, "*.table", False);
fclose(fp);
```

XrtTblWriteResources() requires four arguments. The first two are the table widget and the file pointer.

The third argument is the string to be written before every resource name. It allows you to preserve the resources for several tables in a single file. The string specified by this argument is written before each resource name in the file. For example, if "`*.table`" is specified, *XrtTblWriteResources()* writes the value of `XmNxrtTblNumColumns` as:

```
*.table.xrtTblNumColumns:   6
```

If no string is specified (i.e. the third argument is `NULL`), "`*.`" is written before every resource name.

The fourth argument specifies whether to write the values of inherited resources (such as `XmNwidth`).

Reading Resource Values

To read table resource settings written by *XrtTblWriteResources()*, use *XrtTblReadResources()*:

```
XrtTblReadResources(table, "mytable.res");
```

This procedure applies all table resources to matching tables in the application.

Part II

Reference Appendices

A

Resource Reference

Widget Synopsis ■ *XRT/table Resources*
Constraint Resources ■ *Inherited XRT/table Resources*

This appendix lists all of the XRT/table resources in alphabetical order. Listed after the resource name is its data type, default value and a list of the procedures which may be used with the resource. **C** means it can be defined in a create (e.g. *XtCreateWidget()*), **S** means it can be set (e.g. *XtSetValues()*), and **G** means it can be used in a get (e.g. *XtGetValues()*).

A.1 Widget Synopsis

Include File:	*<Xm/XrtTable.h>*
Class Name:	XtXrtTable
Class Hierarchy:	Core→Composite→Constraint→XmManager→XtXrtTable
Class Pointer:	xtXrtTableWidgetClass
Instantiation:	`widget = XtCreateWidget(name, xtXrtTableWidgetClass...)`
Procedures/Macros:	*XmCreateXrtTable()*, *XtIsXrtTable()*

A.2 XRT/table Resources

XmNxrtTblAlignmentContext **XrtTblAlignment** **XRTTBL_ALIGNMENT_TOPBEGIN** **CSG**

Specifies the alignment of cell and label text, for the current context. Valid values and the positioning they specify are shown below:[1]

XRTTBL_ALIGNMENT_TOPBEGIN	XRTTBL_ALIGNMENT_TOPCENTER	XRTTBL_ALIGNMENT_TOPEND
XRTTBL_ALIGNMENT_MIDDLEBEGIN	XRTTBL_ALIGNMENT_MIDDLECENTER	XRTTBL_ALIGNMENT_MIDDLEEND
XRTTBL_ALIGNMENT_BOTTOMBEGIN	XRTTBL_ALIGNMENT_BOTTOMCENTER	XRTTBL_ALIGNMENT_BOTTOMEND

The inherited `XmNstringDirection` resource specifies which side of the cell is the beginning of text, for use with locales that read right-to-left.

XmNxrtTblAlignmentSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the alignment of text as a series of `XrtTblIntValue` structures.

XmNxrtTblAllowCellResize **XrtTblResize** **XRTTBL_RESIZE_ALL** **CSG**

Specifies how an end-user is able to interactively resize rows/columns. Valid values are: `XRTTBL_RESIZE_ALL`, `XRTTBL_RESIZE_NONE`, `XRTTBL_RESIZE_HORIZONTAL`, and `XRTTBL_RESIZE_VERTICAL`.

XmNxrtTblAllowResize **XrtTblResize** **XRTTBL_RESIZE_NONE** **CSG**

Specifies how the table widget attempts to change its size when resources affecting the size of cells/labels change. When defaulted to `XRTTBL_RESIZE_NONE`, the table attempts to resize only when an application sets `XmNxrtTblVisibleRows` or `XmNxrtTblVisibleColumns`. When set to `XRTTBL_RESIZE_HORIZONTAL`, `XRTTBL_RESIZE_VERTICAL`, or `XRTTBL_RESIZE_ALL`, changes in size-changing cell/label resources (such as `XmNxrtTblMarginWidth`) also force the table to attempt to resize horizontally, vertically, or in both directions. This resource is only useful after a table is realized. Figure 60 on page 123 lists all of the resources that can cause the table to attempt to resize.

XmNxrtTblArgListContext **ArgList** **NULL** **CSG**

Specifies a widget ArgList, for the current context; the last element of the ArgList must be `NULL`. The ArgList values are applied to a widget set in a cell or label before calling any DisplayWidget callbacks. When the resource value is a pointer, it must reference static or allocated memory.

XmNxrtTblArgListSeries **XrtTblSeries *** **NULL** **CSG**

Specifies a widget's ArgList as a series of `XrtTblArgListValue` structures. Unlike other series, these are not copied internally.

---

1. This positioning assumes that `XmNstringDirection` is set to `XmSTRING_DIRECTION_L_TO_R`.

XmNxrtTblBackgroundContext **String** **"Inherited"** **CSG**

Specifies the background color of cells and labels, for the current context. The value can be any color from the X color database or a hexadecimal RGB color value. When set to "Inherited", the color is inherited from Core's `XmNbackground` resource.

When set to `XRTTBL_REPEAT_ROW` or `XRTTBL_REPEAT_COLUMN`, the color is taken from the list specified by `XmNxrtTblRepeatBackgroundColors`, repeating in row or column order.

On a monochrome Server, setting this resource to anything other than black causes white to be used. See section 2.8 on page 29 for details of color specification.

When `XmNxrtTblBackgroundContext` is set, `XmNxrtTblBackgroundPixelContext` automatically changes.

XmNxrtTblBackgroundPixelContext **Pixel** **XRTTBL_NOVALUE** **CSG**

Specifies a cell's background color as a Pixel, for the current context. The resource is ignored if its value is `XRTTBL_NOVALUE`. If this resource is set at creation time, the top and bottom shadow colors will be calculated if not set; when set later, shadow colors do not change[1].

When set to `XRTTBL_REPEAT_ROW_PIXEL` or `XRTTBL_REPEAT_COLUMN_PIXEL`, the color is taken from the list specified by `XmNxrtTblRepeatBackgroundColors`, repeating in row or column order.

If `XmNxrtTblBackgroundPixelContext` and `XmNxrtTblBackgroundContext` are set in the same *XtVaSetValues()* call, the latter is ignored.

Changing `XmNxrtTblBackgroundPixelContext` automatically changes `XmNxrtTblBackgroundContext`. (If the new background color does not correspond to a color name, the value of `XmNxrtTblBackgroundContext` becomes the hexadecimal representation of the color's RGB components.)

XmNxrtTblBackgroundSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the background colors of cells and labels as a series of `XrtTblColorValue` structures.

XmNxrtTblBorderSidesContext **XrtTblBorderSides** **XRTTBL_BORDERSIDE_ALL** **CSG**

Specifies the sides of a cell or label that display the border type in their border area, for the current context. The value is a bitwise OR of the following:

```
XRTTBL_BORDERSIDE_NONE
XRTTBL_BORDERSIDE_ALL
XRTTBL_BORDERSIDE_LEFT
XRTTBL_BORDERSIDE_RIGHT
XRTTBL_BORDERSIDE_TOP
XRTTBL_BORDERSIDE_BOTTOM
```

The width of all cell/label borders is specified by `XmNxrtTblShadowThickness`.

XmNxrtTblBorderSidesSeries **XrtTblSeries *** **NULL** **CSG**

Specifies bordering sides as a series of `XrtTblIntValue` structures. In a resource file, insert a "+" character between symbols, as shown below:

```
*.xrtTblBorderSidesSeries: (ALL ALL BORDERSIDE_LEFT+BORDERSIDE_TOP)
```

1. Border shadow colors are specified by `XmNxrtTblBottomShadowContext` or `XmNxrtTblTopShadowContext`.

XmNxrtTblBorderTypeContext **XrtTblBorderType** **XRTTBL_BORDER_IN** **CSG**

Specifies the style used for cell and label borders, for the current context. Valid values are:

XRTTBL_BORDER_ETCHED_IN	double line; cell appears inset
XRTTBL_BORDER_ETCHED_OUT	double line; cell appears raised
XRTTBL_BORDER_FRAME_IN	1-pixel shadow-in at edge; cell appears framed
XRTTBL_BORDER_FRAME_OUT	1-pixel shadow-out at edge; cell appears framed
XRTTBL_BORDER_IN	cell appears inset
XRTTBL_BORDER_OUT	cell appears raised
XRTTBL_BORDER_PLAIN	shadow drawn in foreground color
XRTTBL_BORDER_NONE	no border

The width of all cell/label borders is specified by XmNxrtTblShadowThickness.

XmNxrtTblBorderTypeSeries **XrtTblSeries *** **NULL** **CSG**

Specifies border type as a series of XrtTblIntValue structures.

XmNxrtTblBottomShadowContext **Pixel** **XRTTBL_NOVALUE** **CSG**

Specifies the bottom shadow color for the current context. When set to XRTTBL_NOVALUE, XRT/table automatically calculates appropriate colors from XmNxrtTblBackgroundContext. This resource must be set to a Motif Pixel color value.

This resource has no corresponding Series resource – XRT/table stores shadow colors in the XmNxrtTblBackgroundSeries.

XmNxrtTblCaseContext **XrtTblCase** **XRTTBL_CASE_AS_IS** **CSG**

Specifies the case of cell values of cells traversed to or edited by the user, for the current context. This resource does not convert the case of existing cell values. XRT/table uses an XmNmodifyVerifyCallback to convert characters typed in the current cell using the toupper and tolower ANSI macros. These macros may not work the same way on all locales. This resource is only used when the cell's data type is string (XmNxrtTblDataTypeContext is XRTTBL_TYPE_STRING). Valid values are:

XRTTBL_CASE_LOWER	Convert to lower case
XRTTBL_CASE_AS_IS	No conversion
XRTTBL_CASE_UPPER	Convert to upper case

XmNxrtTblCaseSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the case conversion as a series of XrtTblIntValue structures.

XmNxrtTblCellValueCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called whenever the table displays an empty cell (a cell with no value in the table's `XrtTblCellValues` structure). The callbacks will be called during initial widget display, scrolling and widget exposure.

Each callback is passed a pointer to an `XrtTblCellValueCallbackStruct`:

```
typedef struct {
    XrtTblReason        reason;                          // Read-only
    XEvent              *event;                          // Read-only
    int                 row, column;                     // Read-only
    XmString            string;
    Boolean             store;                      // initially False
    Boolean             free_string;                // initially False
XrtTblCellValueCallbackStruct;
```

reason is set to `XRTTBL_REASON_CELLVALUE`. *row* and *column* specify the location of the cell. *string* is the cell value returned by your callback routine. *store* should be set to `True` if the string is to be stored in the table; in this case, the callback will not be called again for the cell. If *free_string* is `True`, *string* is freed after being displayed.

By default, *string* accepts values of both XmString and String types (unless you are linking to Motif 2.x). `XmNxrtTblXmStringSetMode` can be used to restrict *string* to values of either type String or type XmString.

If this resource is set, the `XmNxrtTblCellValueFunc` method is ignored.

XmNxrtTblCellValueContext **XmString** **NULL** **CSG**

Specifies a cell value, for the current context. Though its data type is XmString, you can specify a String (if linking to Motif 1.2); XRT/table converts it internally to an XmString using *XmStringCreateLtoR()* with `XmSTRING_DEFAULT_CHARSET`. Strings can also be specified if `XmNxrtTblXmStringSetMode` is set appropriately.

When setting this resource, the cell value is copied into each cell in the context. When getting this resource, the current context must be a single cell.

XmNxrtTblCellValueFunc **XrtTblXmStringProc** **NULL** **CSG**

Specifies a method to be called when displaying a cell that is blank in the table's `XrtTblCellValues` structure. The method is called during initial widget display, scrolling and widget exposure. It has the following prototype:

```
XmString cell_value_func(Widget table, int row, int column)
```

table is an XRT/table widget. The value returned may be either a String or an XmString, depending on the value of `XmNxrtTblXmStringSetMode`. The string returned by the method will be freed by the widget after display.

Use *XtSetValues()* to register this method, not *XtAddCallback()*.

Any method specified by this resource is ignored if `XmNxrtTblCellValueCallback` has been set.

Note: It is *strongly recommended* that `XmNxrtTblCellValueCallback` be used instead of this resource, since `XmNxrtTblCellValueFunc` may not be supported in future versions of XRT/table.

XmNxrtTblCellValues **XrtTblCellValues *** **NULL** **CSG**

Points to an XrtTblCellValues structure containing the cell values. If the structure contains Strings, XRT/table converts each value internally to an XmString using *XmStringCreateLtoR()* with XmSTRING_DEFAULT_CHARSET. Values for a range of cells can be set using *XrtTblSetCellValues()*. Since XrtTblCellValues is optimized, do not retrieve values by directly indexing into it–use the CELL_VALUE macro. See chapter 4 on page 73 for complete information on the cell values structure.

XmNxrtTblCharHeightContext **int** **1** **CSG**

Specifies the number of lines of text that a row can display, for the current context. Multi-line cell values that do not fit in a cell or label are clipped. This resource controls the height only if XmNxrtTblPixelHeightContext is XRTTBL_NOVALUE. When setting this resource, the context should be an entire row (*x*, XRTTBL_ALL). This resource works best with tables that use fixed-width fonts.

When a user interactively resizes a row, this resource is overridden by the pixel height. XmNxrtTblPixelHeightContext updates to specify the pixel height of the user-resized row.

XmNxrtTblCharHeightSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the number of visible text lines in rows as a series of XrtTblIntValue structures.

XmNxrtTblCharWidthContext **int** **10** **CSG**

Specifies the number of characters that a column can display, for the current context. Cell values that do not fit in a cell or label are clipped. This resource controls the width only if XmNxrtTblPixelWidthContext is XRTTBL_NOVALUE. When setting this resource, the context should be an entire column (XRTTBL_ALL, *x*). This resource works best with tables that use fixed-width fonts.

When a user interactively resizes a column, this resource is overridden by the pixel width. XmNxrtTblPixelWidthContext updates to specify the pixel width of the user-resized column.

XmNxrtTblCharWidthSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the number of visible characters in columns as a series of XrtTblIntValue structures.

XmNxrtTblClipChildren **Widget *** **NULL** **G**

Specifies a list of the table's private area-clipping child widgets; the last element of the list must be NULL. Translations and event handlers must be set on these widgets rather than on the table itself.

XmNxrtTblClipChildrenPersistent **Boolean** **False** **CSG**

Specifies whether the Clip children (see XmNxrtTblClipChildren) are permanently created. If False, Clip children are created and destroyed as necessary. If you are placing XRT/table in a popup shell that could be popped down before the XRT/table widget is destroyed, it is recommended that you set the resource XmNxrtTblClipChildrenPersistent to True.

XmNxrtTblClipPixmapContext **Pixmap** **XmUNSPECIFIED_PIXMAP** **CSG**

Specifies the clip pixmap (clip mask) to be used when drawing a pixmap, for the current context. (The pixmap itself is specified by the `XmNxrtTblPixmapContext` resource; the clip pixmap controls how much of this pixmap is to be displayed.)

XmNxrtTblClipPixmapSeries **XrtTblSeries *** **NULL** **CSG**

Specifies clip pixmaps as a series of `XrtTblPixmapValue` structures.

XmNxrtTblColumnLabelOffset **int** **0** **CSG**

Specifies the spacing between the column labels and the cell area in pixels. If a negative value is specified, the labels will overlap the cell area.

XmNxrtTblColumnLabelPlacement **XrtTblRowPlacement** **XRTTBL_PLACE_TOP** **CSG**

Specifies the position of the column labels on the screen. Valid values are `XRTTBL_PLACE_TOP` and `XRTTBL_PLACE_BOTTOM`.

XmNxrtTblColumnLabels **XrtTblXmStringList *** **NULL** **CSG**

Specifies all of the column labels. Though its data type is XmString, you can specify a String (if linking to Motif 1.2); XRT/table converts it internally to an XmString. Strings can also be specified if `XmNxrtTblXmStringSetMode` is set appropriately.

XmNxrtTblContext **XrtTblContext *** **ALL ALL** **CSG**

Specifies the current context unless `XmNxrtTblRangeContext` has been set. Subsequent Context resource settings are applied to this context.[1] When this resource is set to a valid context, `XmNxrtTblRangeContext` is set to `NULL` as a side-effect.

A context is specified as a *row and column index*. The following special row/column values specify other parts of the table:

`XRTTBL_ALL`	all cells and labels in a row or column
`XRTTBL_LABEL`	a row or column label
`XRTTBL_ALLCELLS`	all cells (but not labels) in a row or column

If both *row* and *column* are `XRTTBL_ALL`, the context is set to the entire table (all rows, columns, and labels). Some context examples:

`1 2`	cell (1, 2)
`XRTTBL_ALL 1`	all cells and the label of column 1
`2 XRTTBL_ALL`	all cells and the label of row 2
`XRTTBL_ALL` `XRTTBL_LABEL`	all row labels

1. See section 2.4 on page 25 for more information on contexts and Context resources.

A context can alternately be set to a range using `XmNxrtTblRangeContext`.

Use *XrtTblSetContext()* to set this resource. `XmNxrtTblContext` cannot be set in a resource file.

XmNxrtTblContextValueCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called every time a cell or label context value is needed during size calculations or drawing. Each routine is passed a pointer to an `XrtTblContextValueCallbackStruct`:

```
typedef struct {
    int                     reason;                         // Read-only
    int                     row, column;                    // Read-only
    XEvent                  *event;                         // Read-only
    XrtTblAction            action;                         // Read-only
    XrtTblContextReason     context_reason;                 // Read-only
    XrtTblContextValue      value;
} XrtTblContextValueCallbackStruct;

typedef union {
    XrtTblAlignment         alignment;
    ArgList                 arglist;
    Pixel                   background;
    XrtTblBorderSides       border_sides;
    XrtTblBorderType        border_type;
    Pixel                   bottom_shadow;
    int                     char_height;
    int                     char_width;
    Pixmap                  clip_pixmap;
    XrtTblDataType          datatype;
    Boolean                 editable;
    ArgList                 field_arglist;
    Pixel                   foreground;
    int                     maxlength;
    Boolean                 multiline;
    XtPointer               picklist;
    int                     pixel_height;
    int                     pixel_width;
    Pixmap                  pixmap;
    XrtTblPixmapLayout      pixmap_layout;
    Boolean                 pixmap_shadow;
    String                  ps_fontlist;
    Boolean                 sensitive;
    Pixel                   top_shadow;
    XtPointer               userdata;
    Boolean                 widget_shadow;
    Boolean                 wordwrap;
} XrtTblContextValue;
```

reason is always `XRTTBL_REASON_CONTEXTVALUE`. *event* is `NULL`. *action* is `XRTTBL_ACTION_CALCULATE` or `XRTTBL_ACTION_DRAW`. *context_reason* specifies which value is desired. *value* is the value that will be used for the applicable context; it is obtained from the series, or from the CellValue/LabelCreate callback. The value will not be freed after use, and must point to static storage.

context_reason is one of the following:

XRTTBL_CONTEXT_ALIGNMENT	XRTTBL_CONTEXT_FOREGROUND
XRTTBL_CONTEXT_ARGLIST	XRTTBL_CONTEXT_MAXLENGTH
XRTTBL_CONTEXT_BACKGROUND	XRTTBL_CONTEXT_MULTILINE
XRTTBL_CONTEXT_BORDERSIDES	XRTTBL_CONTEXT_PIXEL_HEIGHT
XRTTBL_CONTEXT_BORDERTYPE	XRTTBL_CONTEXT_PIXEL_WIDTH
XRTTBL_CONTEXT_BOTTOMSHADOW	XRTTBL_CONTEXT_PIXMAP
XRTTBL_CONTEXT_CASE	XRTTBL_CONTEXT_PIXMAP_LAYOUT
XRTTBL_CONTEXT_CHAR_HEIGHT	XRTTBL_CONTEXT_PIXMAP_SHADOW
XRTTBL_CONTEXT_CHAR_WIDTH	XRTTBL_CONTEXT_PS_FONTLIST
XRTTBL_CONTEXT_CLIP_PIXMAP	XRTTBL_CONTEXT_SENSITIVE
XRTTBL_CONTEXT_CURRENCYPICKLIST	XRTTBL_CONTEXT_STRINGPICKLIST
XRTTBL_CONTEXT_DATATYPE	XRTTBL_CONTEXT_TOP_SHADOW
XRTTBL_CONTEXT_DATEPICKLIST	XRTTBL_CONTEXT_USERDATA
XRTTBL_CONTEXT_EDITABLE	XRTTBL_CONTEXT_WIDGET_SHADOW
XRTTBL_CONTEXT_FIELD_ARGLIST	XRTTBL_CONTEXT_WORDWRAP
XRTTBL_CONTEXT_FLOATPICKLIST	

XmNxrtTblCreateWidgetCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application before and after creating/cloning a widget in a cell/label. A pointer to an `XrtTblCreateWidgetCallbackStruct` is passed to each callback. See section 6.1.2 on page 115 for complete details on using this resource. The form of the callback is:

```
void
CallbackProcedure(
    Widget      table,
    XtPointer   client_data,
    XtPointer   call_data    // points to an
    )                        // XrtTblCreateWidgetCallbackStruct
```

where `XrtTblCreateWidgetCallbackStruct` is defined as:

```
typedef struct {
    XrtTblReason        reason;                              // Read-only
    XEvent              *event;                              // Read-only
    int                 row, column;                         // Read-only
    Widget              parent;                              // Read-only
    WidgetClass         widget_class;                        // Read-only
    String              class_name, name;                    // Read-only
    Widget              clone_source;                        // Read-only
    Widget              new_widget;
} XrtTblCreateWidgetCallbackStruct;
```

reason is either `XRTTBL_REASON_CREATE_WIDGET_BEGIN` or `XRTTBL_REASON_CREATE_WIDGET_END`; *row, column* specify the location of the cell or label to contain the widget; *parent* is the parent widget; *widget_class* is the class pointer of the widget (`NULL` when XRT/table does not recognize

the widget); *class_name* and *name* specify the class and instance name of the widget; *clone_source* is the widget being cloned (`NULL` if widget is not a clone); *new_widget* is the widget may be set (only meaningful when reason is `XRTTBL_REASON_CREATE_WIDGET_END`).

XmNxrtTblCurrencyPickListContext **double *** **NULL** **CSG**

Specifies a list of suggested values for the current context. The value of the first element must be the remaining number of elements in the list.

XmNxrtTblCurrencyPickListSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the pick list as a series of `XrtTblPickListValue` structures.

XmNxrtTblCursorCorner **Cursor** **XC_bottom_right_corner** **CSG**

Specifies the mouse cursor displayed at the bottom-right corner of a cell/label when `XmNxrtTblAllowCellResize` is `XRTTBL_RESIZE_ALL`. The default cursor is created from the `XC_bottom_right_corner` symbol.

XmNxrtTblCursorHorizontal **Cursor** **XC_right_side** **CSG**

Specifies the mouse cursor displayed at the right edge of a cell/label when `XmNxrtTblAllowCellResize` is `XRTTBL_RESIZE_ALL` or `XRTTBL_RESIZE_HORIZONTAL`. The default cursor is created from the `XC_right_side` symbol.

XmNxrtTblCursorNontraversable **Cursor** **XC_plus** **CSG**

Specifies the mouse cursor displayed inside a non-traversable cell (`XmNxrtTblTraversableContext` is `False`). The default cursor is created from the `XC_plus` symbol.

XmNxrtTblCursorTraversable **Cursor** **XC_xterm** **CSG**

Specifies the mouse cursor displayed inside a traversable cell (`XmNxrtTblTraversableContext` is `True`). The default cursor is created from the `XC_xterm` symbol.

XmNxrtTblCursorVertical **Cursor** **XC_bottom_side** **CSG**

Specifies the mouse cursor displayed at the bottom edge of a cell/label when `XmNxrtTblAllowCellResize` is `XRTTBL_RESIZE_ALL` or `XRTTBL_RESIZE_VERTICAL`. The default cursor is created from the `XC_bottom_side` symbol.

XmNxrtTblDataTypeContext **XrtTblDataType** **XRTTBL_TYPE_STRING** **CSG**

Specifies the data type of a cell/label value, for the current context. When set, this resource affects cell and/or label value changes from that point on. Before committing a changed cell/label value to the cell or label values structures,[1] XRT/table uses a ValidateCell callback to call a data validation method for the data type specified by this resource. This resource can be set to any data type specified by the `XmNxrtTblDataTypeList` resource.

1. `XmNxrtTblValidatePolicy` determines when to check cell/label value changes against their data type.

If an XmText widget is used to edit cell values, valid built-in data types are: XRTTBL_TYPE_STRING, XRTTBL_TYPE_DOUBLE, XRTTBL_TYPE_INT, XRTTBL_TYPE_FLOAT, and XRTTBL_TYPE_BOOLEAN.

If an XRT/field widget is used to edit cell values, valid data types are:

XRTTBL_TYPE_FIELD_CURRENCY	XrtCurrencyField
XRTTBL_TYPE_FIELD_DATE	XrtDateField
XRTTBL_TYPE_FIELD_FLOAT	XrtFloatField
XRTTBL_TYPE_FIELD_INT	XrtIntField
XRTTBL_TYPE_FIELD_STRING	XrtStringField
XRTTBL_TYPE_FIELD_COMBO_CURRENCY	XrtComboBox
XRTTBL_TYPE_FIELD_COMBO_DATE	XrtComboBox
XRTTBL_TYPE_FIELD_COMBO_FLOAT	XrtComboBox
XRTTBL_TYPE_FIELD_COMBO_INT	XrtComboBox
XRTTBL_TYPE_FIELD_COMBO_STRING	XrtComboBox
XRTTBL_TYPE_FIELD_SPIN_CURRENCY	XrtSpinBox
XRTTBL_TYPE_FIELD_SPIN_DATE	XrtSpinBox
XRTTBL_TYPE_FIELD_SPIN_FLOAT	XrtSpinBox
XRTTBL_TYPE_FIELD_SPIN_INT	XrtSpinBox
XRTTBL_TYPE_FIELD_SPIN_STRING	XrtSpinBox

An XmText widget is displayed if *XrtTblEnableXrtField()* has not been previously called.

XmNxrtTblDataTypeList **XrtTblDataTypeRoutine *** ***see below*** **CSG**

Specifies a list of data types and validation methods; the last element of the list must be NULL. The data types specified are the valid types that can be set for cells/labels (using XmNxrtTblDataTypeContext). The methods specified are called by XRT/table to validate that data type.

This resource takes an XrtTblDataTypeRoutine, defined as:

```
typedef struct {
    XrtTblDataType      type;
    XtCallbackProc      routine;
} XrtTblDataTypeRoutine;
```

The default data type list is as follows:

XRTTBL_TYPE_STRING	NULL (no validation method)

XRTTBL_TYPE_DOUBLE	*XrtTblCheckIsDouble()*
XRTTBL_TYPE_INT	*XrtTblCheckIsInt()*
XRTTBL_TYPE_FLOAT	*XrtTblCheckIsFloat()*
XRTTBL_TYPE_BOOLEAN	*XrtTblCheckIsBoolean()*
XRTTBL_TYPE_FIELD_CURRENCY	uses XrtCurrencyField validations
XRTTBL_TYPE_FIELD_DATE	uses XrtDateField validations
XRTTBL_TYPE_FIELD_FLOAT	uses XrtFloatField validations
XRTTBL_TYPE_FIELD_INT	uses XrtIntField validations
XRTTBL_TYPE_FIELD_STRING	uses XrtStringField validations
XRTTBL_TYPE_FIELD_COMBO_CURRENCY	uses XrtCurrencyField validations
XRTTBL_TYPE_FIELD_COMBO_DATE	uses XrtDateField validations
XRTTBL_TYPE_FIELD_COMBO_FLOAT	uses XrtFloatField validations
XRTTBL_TYPE_FIELD_COMBO_INT	uses XrtIntField validations
XRTTBL_TYPE_FIELD_COMBO_STRING	uses XrtStringField validations
XRTTBL_TYPE_FIELD_SPIN_CURRENCY	uses XrtCurrencyField validations
XRTTBL_TYPE_FIELD_SPIN_DATE	uses XrtDateField validations
XRTTBL_TYPE_FIELD_SPIN_FLOAT	uses XrtFloatField validations
XRTTBL_TYPE_FIELD_SPIN_INT	uses XrtIntField validations
XRTTBL_TYPE_FIELD_SPIN_STRING	uses XrtStringField validations
XRTTBL_TYPE_LAST	Index your own validation methods from this enumeration. Use XmNxrtTblDataTypeList to add data types and validation methods.

When setting this resource, new data types are added to the list (or replaced if the data type exists). New data types should have enumerated values greater than XRTTBL_TYPE_LAST. If the method for a given data type is NULL, no validation will be performed for the type.

XmNxrtTblDataTypeSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the data type of each cell as a series of XrtTblIntValue structures.

XmNxrtTblDatePickListContext **struct tm **** **NULL** **CSG**

Specifies a list of suggested values for the current context. The list of values must be NULL terminated. The table makes an internal copy of the values passed.

XmNxrtTblDatePickListSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the pick list as a series of `XrtTblPickListValue` structures.

XmNxrtTblDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be output to *stderr* from the Xt error database. When `False`, only resource conversion errors are output.

XmNxrtTblDisplayClipArrows **Boolean** **True** **CSG**

Specifies whether to display clipping arrows when a cell/label value cannot be fully displayed in its cell/label. Clipping arrows appear at the bottom-right corner of a cell/label.

XmNxrtTblDisplayHorizScrollBar **XrtTblDispScrollBar** **XRTTBL_DISPSB_AS_NEEDED** **CSG**

Specifies when to display the horizontal scrollbar. Valid values are:

`XRTTBL_DISPSB_ALWAYS`	scrollbar is always displayed
`XRTTBL_DISPSB_NEVER`	scrollbar is never displayed
`XRTTBL_DISPSB_AS_NEEDED`	scrollbar is displayed when the number of columns in the scrollable region exceeds the number of visible columns

XmNxrtTblDisplayVertScrollBar **XrtTblDispScrollBar** **XRTTBL_DISPSB_AS_NEEDED** **CSG**

Specifies when to display the vertical scrollbar. Valid values are:

`XRTTBL_DISPSB_ALWAYS`	scrollbar is always displayed
`XRTTBL_DISPSB_NEVER`	scrollbar is never displayed
`XRTTBL_DISPSB_AS_NEEDED`	scrollbar is displayed when the number of rows in the scrollable region exceeds the number of visible rows.

XmNxrtTblDisplayWidgetCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called before drawing a widget in a cell or label. DisplayWidget callbacks are triggered when the table is initially displayed and when the cell is scrolled into view.[1] A pointer to an `XrtTblDisplayWidgetCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      table,
    XtPointer   client_data,
    XtPointer   call_data   // points to an XrtTblDisplayWidgetCallbackStruct
```

1. If a widget ArgList has been defined for a context (using `XmNxrtTblArgListContext`), it is applied to the widget before calling DisplayWidget callbacks.

where XrtTblDisplayWidgetCallbackStruct is defined as:

```
typedef struct {
    XrtTblReason        reason;                                 // Read-only
    XEvent              *event;                                 // Read-only
    int                 row, column;                            // Read-only
    Widget              clone_source;                           // Read-only
    Widget              widget;
} XrtTblDisplayWidgetCallbackStruct;
```

reason is always XRTTBL_REASON_DISPLAY_WIDGET; *row*, *column* specify the location of the cell or label containing the widget; *clone_source* is the widget being cloned (NULL if the widget is not cloned); *widget* may be set.

If the widget is being cloned, callbacks must be added to the XmNxrtTblDisplayWidgetCallback list before the widget location is specified (by XmNxrtTblWidgetLocation or XmNxrtTblWidgetContext), as the widget is cloned when the location is specified. (This behavior differs from that of previous releases of XRT/table.)

XmNxrtTblDoubleBuffer **XrtTblDoubleBuffer** **XRTTBL_DOUBLEBUFFER_CELL** **CSG**

Controls which double-buffering technique is used when displaying and updating the table. Valid values are:

XRTTBL_DOUBLEBUFFER_CELL	double-buffer on a cell/label basis
XRTTBL_DOUBLEBUFFER_WINDOW	double-buffer entire window
XRTTBL_DOUBLEBUFFER_NONE	disable double-buffering

When double-buffering is enabled, display updates are first drawn to an off-screen area, then copied to the screen.

XmNxrtTblEditableContext **Boolean** **True** **CSG**

Specifies whether cell values can be changed by an end-user, for the current context. A cell that is not editable (but is traversable) maps the XmText widget over it when traversed to, but the terminal beeps if the user attempts to change the value. For a cell to be editable, it must also have XmNxrtTblTraversableContext set to True.

Setting this resource to True for a row/column label context ensures that a row/column can be dropped into the table at that location.

XmNxrtTblEditableSeries **XrtTblSeries *** **NULL** **CSG**

Specifies cell editability as a series of XrtTblBooleanValue structures.

XmNxrtTblEnterCellCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called before and after completing a traversal from one cell to another. EnterCell callbacks are triggered by the *TraverseToCell()* action. A pointer to an XrtTblEnterCellCallbackStruct is passed to each callback. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      table,
    XtPointer   client_data,
    XtPointer   call_data       // points to an XrtTblEnterCellCallbackStruct
    )
```

where `XrtTblEnterCellCallbackStruct` is defined as:

```
typedef struct {
    XrtTblReason    reason;         // Read-only
    XEvent          *event;         // Read-only
    String          *params;        // Read-only
    Cardinal        num_params;     // Read-only
    int             row, column;    // Read-only
    Boolean         doit;
} XrtTblEnterCellCallbackStruct;
```

reason is either `XRTTBL_REASON_ENTER_CELL_BEGIN` or `XRTTBL_REASON_ENTER_CELL_END`; *row, column* specify the cell being traversed to; *doit* is a flag that determines whether to allow the cell to be edited.

XmNxrtTblExportCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called before and after an export (drag) operation has been performed. A pointer to an `XrtTblExportCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
   Widget      table,
   XtPointer   client_data,
   XtPointer   call_data       // points to an XrtTblExportCallbackStruct
   )
```

where `XrtTblExportCallbackStruct` is defined as:

```
typedef struct {
    XrtTblReason    reason;         // Read-only
    XEvent          *event;         // Read-only
    int             row, column;    // Read-only
    String          target;         // Read-only
    Boolean         internal;       // Read-only
    XrtTblRange     range;
    unsigned char   operation;
    Boolean         doit;
    Boolean         bell;
} XrtTblExportCallbackStruct;
```

reason is one of `XRTTBL_REASON_EXPORT_DRAG_BEGIN` (start of drag – mouse button depressed), `XRTTBL_REASON_EXPORT_TARGET` (drop about to start), or `XRTTBL_REASON_EXPORT_DRAG_END` (end of drag and drop – mouse button up); *event* is `NULL`; *row* and *column* represent the location

of the cell containing the cursor. *internal* is `True` if the drag and drop operation is from one part of a table to another part of the same table.

target describes the data being exported (the drag target). If *reason* is `XRTTBL_REASON_EXPORT_DRAG_BEGIN` or `XRTTBL_REASON_EXPORT_DRAG_END`, *target* is one of the following:

`XRTTBL_TGT_RANGE`	drag target is a range of cells
`XRTTBL_TGT_COLUMNS`	drag target is one or more table columns
`XRTTBL_TGT_ROWS`	drag target is one or more table rows
`XRTTBL_TGT_TEXT`	drag target is an unspecified character string (e.g., not from a table)

If *reason* is `XRTTBL_REASON_EXPORT_TARGET`, *target* is a target in the list specified by `XmNxrtTblExportTargets`. (The callbacks are called once for each target for which a converter is defined. If no converters are defined, a default converter is used, and the callbacks are called when the default converter is about to perform its conversion.)

range is the range of cells/labels to be dragged. *operation* is either `XmDROP_COPY` (copy the selected range) or `XmDROP_MOVE` (move the selected range, destroying the existing copy). *doit* allows you to override whether the drag and drop operation is to take place. If *bell* is set to `True`, the terminal beeps.

XmNxrtTblExportDelimiter **String** **"\t"** **CSG**

Specifies the character string to use as a delimiter between cells when exporting (dragging) normal text from a table. To specify that no delimiter is to be written, set this resource to `XRTTBL_DELIMIT_NONE`. To use the default, set this resource to `XRTTBL_DELIMIT_DEFAULT`.

`XRTTBL_DELIMIT_BY_WHITESPACE` indicates that any combination of spaces and tabs is treated as a delimiter.

XmNxrtTblExportTargets **XrtTblImportExportStruct ** *see below*** **CSG**

Specifies a list of items (targets) that can be exported from a table. Each element of the list must be of type `XrtTblImportExportStruct`:

```
typedef struct {
    String          target;
    XrtTblImportExportConvertProc converter;
} XrtTblImportExportStruct;
```

The last element of this list must be `NULL`.

target provides information on the data being exported. When a drag operation is performed, an `XrtTblImportExportStruct` element is automatically created with one of the following *target* values:

`XRTTBL_TGT_RANGE`	ranges of cells
`XRTTBL_TGT_COLUMNS`	entire columns
`XRTTBL_TGT_ROWS`	entire rows

If you are exporting to an XmText/XmTextField widget, you need to include the following targets:

```
XRTTBL_TGT_COMPOUND_TEXT
XRTTBL_TGT_TEXT
XRTTBL_TGT_STRING
```

When text is exported to an XmText/XmTextField widget, the string specified by `XmNxrtTblExportDelimiter` appears between the text from neighboring cells.

The following targets can also be specified:

`XRTTBL_TGT_CELLS`	export data in XRT/table data transfer format
`XRTTBL_TGT_ROWLABELS`	include row labels when dragging
`XRTTBL_TGT_COLUMNLABELS`	include column labels when dragging

converter defines the conversion method to use when dragging a particular type of data; when `NULL`, the default conversion method for the data type is used. Converters should not be defined for `XRTTBL_TGT_RANGE`, `XRTTBL_TGT_COLUMNS`, `XRTTBL_TGT_ROWS`, `XRTTBL_TGT_ROWLABELS` or `XRTTBL_TGT_COLUMNLABELS`.

`XRTTBL_TGT_ROWLABELS` and `XRTTBL_TGT_COLUMNLABELS` are not useful unless `XRTTBL_TGT_CELLS` is also specified.

The default for `XmNxrtTblExportTargets` is a list containing the following targets:

```
XRTTBL_TGT_COMPOUND_TEXT
XRTTBL_TGT_TEXT
XRTTBL_TGT_STRING
XRTTBL_TGT_CELLS
```

XmNxrtTblExposeCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application before and after repainting some portion of the table. Expose callbacks are triggered whenever the widget redraws or modifies an area of its screen image (row labels, column labels, frozen rows, etc.). A pointer to an`XrtTblExposeCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget          table,
    XtPointer       client_data,
    XtPointer       call_data   // points to an XrtTblExposeCallbackStruct
    )
```

where `XrtTblExposeCallbackStruct` is defined as:

```
typedef struct {
    XrtTblReason        reason;     // Read-only
    XEvent              *event;     // Read-only
    XrtTblRange         range;      // Read-only
} XrtTblExposeCallbackStruct;
```

reason is either `XRTTBL_REASON_EXPOSE_BEGIN` or `XRTTBL_REASON_EXPOSE_END`; *event* is `NULL` if the event was internally-generated; *range* specifies the range affected by the repaint.

XmNxrtTblFieldArgListContext **ArgList** **NULL** **CSG**

Specifies a list of Args, for the current context; the last element of the list must be `NULL`. The args are applied only when the cell contains an XRT/field widget (that is, when an XRT/field `XmNxrtTblDataTypeContext` has been set). Note that this resource is not copied internally.

XmNxrtTblFieldArgListSeries **XrtTblSeries *** **NULL** **CSG**

Specifies XRT/field resources for a cell as a series of `XrtTblArgListValue` structures. Note that this resource is not copied internally.

XmNxrtTblFloatPickListContext **double *** **NULL** **CSG**

Specifies a list of suggested values for the current context. The value of the first element must be the remaining number of elements in the list. The table makes an internal copy of the values passed.

XmNxrtTblFloatPickListSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the pick list as a series of `XrtTblPickListValue` structures.

XmNxrtTblFontListContext **XmFontList** **“fixed”** **CSG**

Specifies the font list to be used to display cell and/or label values, for the current context.

XmNxrtTblFontListSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the font list for cells and/or labels as a series of `XrtTblFontListValue` structures.

XmNxrtTblForegroundContext **String** **“Inherited”** **CSG**

Specifies the foreground color of cells and labels, for the current context. The value can be any color from the X color database or a hexadecimal RGB color value. When set to “Inherited”, the color is inherited from XmManager’s `XmNforeground` resource.

When set to `XRTTBL_REPEAT_ROW` or `XRTTBL_REPEAT_COLUMN`, the color is taken from the list specified by `XmNxrtTblRepeatForegroundColors`, repeating in row or column order.

On a monochrome Server, setting this resource to anything other than white causes black to be used. See section 2.8 on page 29 for details of color specification.

When `XmNxrtTblForegroundContext` is set, `XmNxrtTblForegroundPixelContext` automatically changes.

XmNxrtTblForegroundPixelContext **Pixel** **XRTTBL_NOVALUE** **CSG**

Specifies a cell's foreground color as a Pixel, for the current context. The resource is ignored if its value is `XRTTBL_NOVALUE`. If this resource is set at creation time, the top and bottom shadow colors will be calculated if not set; when set later, shadow colors do not change[1].

When set to `XRTTBL_REPEAT_ROW_PIXEL` or `XRTTBL_REPEAT_COLUMN_PIXEL`, the color is taken from the list specified by `XmNxrtTblRepeatForegroundColors`, repeating in row or column order.

If `XmNxrtTblForegroundPixelContext` and `XmNxrtTblForegroundContext` are set in the same *XtVaSetValues()* call, the latter is ignored.

Changing `XmNxrtTblForegroundPixelContext` automatically changes `XmNxrtTblForegroundContext`. (If the new foreground color does not correspond to a color name, the value of `XmNxrtTblForegroundContext` becomes the hexadecimal representation of the color's RGB components.)

XmNxrtTblForegroundSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the foreground colors of cells and labels as a series of `XrtTblColorValue` structures.

XmNxrtTblFrameBorderType **XrtTblBorderType** **XRTTBL_BORDER_NONE** **CSG**

Specifies the type of border for the frame surrounding the cell and label areas. Valid border types are:

`XRTTBL_BORDER_NONE`	no border
`XRTTBL_BORDER_ETCHED_IN`	double line; cell appears inset
`XRTTBL_BORDER_ETCHED_OUT`	double line; cell appears raised
`XRTTBL_BORDER_IN`	cell appears inset
`XRTTBL_BORDER_OUT`	cell appears raised
`XRTTBL_BORDER_PLAIN`	shadow drawn in foreground color

XmNxrtTblFrameShadowThickness **Dimension** **0** **CSG**

Specifies the thickness of the frame shadows surrounding the cell and label areas.

XmNxrtTblFrozenColumnPlacement **XrtTblColumnPlacement** **XRTTBL_PLACE_LEFT** **CSG**

Specifies the location of all frozen columns within the widget display. Changing the placement of frozen columns does not change the location of the columns in the `XrtTblCellValues` structure. Valid values are `XRTTBL_PLACE_LEFT` and `XRTTBL_PLACE_RIGHT`.

1. Border shadow colors are specified by `XmNxrtTblBottomShadowContext` or `XmNxrtTblTopShadowContext`.

XmNxrtTblFrozenColumns **int** **0** **CSG**

Specifies the number of columns from the start of the table that are not horizontally scrollable. Placement of the frozen columns on the screen is specified by `XmNxrtTblFrozenColumnPlacement`.

XmNxrtTblFrozenRowPlacement **XrtTblRowPlacement** **XRTTBL_PLACE_TOP** **CSG**

Specifies the location of all frozen rows within the widget display. Changing the placement of frozen rows does not change the location of the rows in the `XrtTblCellValues` structure. Valid values are `XRTTBL_PLACE_TOP` and `XRTTBL_PLACE_BOTTOM`.

XmNxrtTblFrozenRows **int** **0** **CSG**

Specifies the number of rows from the start of the table that are not vertically scrollable. Placement of the frozen rows on the screen is specified by `XmNxrtTblFrozenRowPlacement`.

XmNxrtTblHorizontalScrollBar **Widget** **NULL** **SG**

Specifies the widget ID of the horizontal scrollbar. Do not set to `NULL` if you want to disable display of the scrollbar—use `XmNxrtTblDisplayHorizScrollBar`.

XmNxrtTblHorizScrollBarLeftAttachment **XrtTblAttachment** **XRTTBL_ATTACH_CELLS** **CSG**

Specifies how the left end of the horizontal scrollbar is attached to the table. When set to `XRTTBL_ATTACH_CELLS`, the scrollbar ends at the left edge of the visible cells. When set to `XRTTBL_ATTACH_SIDE`, the scrollbar ends at the left edge of the widget.

XmNxrtTblHorizScrollBarPlacement **XrtTblRowPlacement** **XRTTBL_PLACE_BOTTOM** **CSG**

Specifies the side of the table to place the horizontal scrollbar. When set to `XRTTBL_PLACE_BOTTOM`, the scrollbar is placed below the table. When set to `XRTTBL_PLACE_TOP`, the scrollbar is placed above the table.

XmNxrtTblHorizScrollBarPosition **XrtTblSBPosition** **XRTTBL_SBPOSITION_CELLS** **CSG**

Specifies how the horizontal scrollbar is attached to the table. When set to `XRTTBL_SBPOSITION_CELLS`, the scrollbar is attached to the cell/label viewport. When set to `XRTTBL_SBPOSITION_SIDE`, the scrollbar is attached to the side of the widget. `XmNxrtTblSpace` specifies the distance between the scrollbar and the table.

XmNxrtTblHorizScrollBarRightAttachment **XrtTblAttachment** **XRTTBL_ATTACH_CELLS** **CSG**

Specifies how the right end of the horizontal scrollbar is attached to the table. When set to `XRTTBL_ATTACH_CELLS`, the scrollbar ends at the right edge of the visible cells. When set to `XRTTBL_ATTACH_SIDE`, the scrollbar ends at the right edge of the widget.

XmNxrtTblImportCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called before and after an import (drop) operation has been performed. A pointer to an `XrtTblImportCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget          table,
    XtPointer       client_data,
    XtPointer       call_data       // points to an XrtTblImportCallbackStruct
        )
```

where XrtTblImportCallbackStruct is defined as:

```
    typedef struct {
        int             reason;         // Read-only
        XEvent          *event;         // Read-only
        int             row, column;    // Read-only
        String          target;         // Read-only
        Boolean         internal;       // Read-only
        unsigned char   operation;
        unsigned char   dropSiteStatus;
        Boolean         insert;
        Boolean         doit;
        Boolean         bell;
    } XrtTblImportCallbackStruct;
```

reason is one of XRTTBL_REASON_IMPORT_DRAG_MOTION (drop site informed that drag is occurring), XRTTBL_REASON_IMPORT_BEGIN (mouse button released), or XRTTBL_REASON_IMPORT_TARGET (drop conversion about to start); *event* is NULL; *row* and *column* represent the location of the cell containing the cursor. *internal* is True if the drag and drop operation is from one part of a table to another part of the same table.

target describes the data being imported (the drop target). If *reason* is XRTTBL_REASON_IMPORT_BEGIN or XRTTBL_REASON_IMPORT_DROP_END, *target* is one of the following:

XRTTBL_TGT_RANGE	drop target is a range of cells
XRTTBL_TGT_COLUMNS	drop target is one or more table columns
XRTTBL_TGT_ROWS	drop target is one or more table rows
XRTTBL_TGT_TEXT	drop target is an unspecified character string (e.g., not from a table)

target can only be modified if *reason* is XRTTBL_REASON_IMPORT_BEGIN and you are changing from XRTTBL_TGT_COLUMNS or XRTTBL_TGT_ROWS to XRTTBL_TARGET_RANGE.

If *reason* is XRTTBL_REASON_IMPORT_TARGET, *target* is a target in the list specified by XmNxrtTblImportTargets. (The callbacks are called once for each target for which a converter is defined. If no converters are defined, a default converter is used, and the callbacks are called when the default converter is about to perform its conversion.)

operation is either XmDROP_COPY (copy the selected range) or XmDROP_MOVE (move the selected range, destroying the existing copy). *dropSiteStatus* is either XmVALID_DROP_SITE (drop valid at this site; the default) or XmINVALID_DROP_SITE (drop invalid). *insert* specifies whether the dropped data is to be inserted into the table at the specified location, if possible; *insert* is either True (insert if possible; the default) or False (overwrite existing data), and can be changed. *doit* allows you to override whether the drag and drop operation is to take place. If *bell* is set to True, the terminal beeps.

XmNxrtTblImportDelimiter **String** **"\t"** **CSG**

Specifies the character string to use as a delimiter between cells when importing (dropping) normal text into a table. To specify that no delimiter is to be assumed, set this resource to XRTTBL_DELIMIT_NONE. To use the default, set this resource to XRTTBL_DELIMIT_DEFAULT.

XRTTBL_DELIMIT_BY_WHITESPACE indicates that any combination of spaces and tabs is treated as a delimiter.

XmNxrtTblImportTargets **XrtTblImportExportStruct ** *see below*** **CSG**

Specifies a list of items (targets) that can be imported into a table. Each element of the list must be of type XrtTblImportExportStruct:

```
typedef struct {
    String          target;
    XrtTblImportExportConvertProc converter;
} XrtTblImportExportStruct;
```

The last element of this list must be NULL.

Each item in the target list specifies a type of data that can be imported; *target* is used to specify this data type. When importing from another (or the same) table, target items can contain any or all of the following *target* values:

XRTTBL_TGT_RANGE	accept ranges of cells
XRTTBL_TGT_COLUMNS	accept entire columns
XRTTBL_TGT_ROWS	accept entire rows

The following target items specify whether labels and/or cells are to be dropped:

XRTTBL_TGT_CELLS	text is in XRT/table data transfer format
XRTTBL_TGT_ROWLABELS	include row labels when dropping
XRTTBL_TGT_ROWLABELS	include column labels when dropping

XRTTBL_TGT_ROWLABELS and XRTTBL_TGT_COLUMNLABELS cannot be specified unless XRTTBL_TGT_CELLS is also specified.

To ensure safe and efficient operation when dragging and dropping rows or columns within a table, add the following target item to the list:

XRTTBL_TGT_INTERNALMOVE specifies high-speed internal drag and drop (within a single table)

When this target is added, overlapping drag and drop operations behave properly. In internal drag and drop operations, everything in a row or column is moved, including fontlists and pixmaps.

If you are importing from an XmText widget, define any or all of the following targets:

```
XRTTBL_TGT_COMPOUND_TEXT
XRTTBL_TGT_TEXT
XRTTBL_TGT_STRING
```

The actual data type needed depends on how the data was dragged from the XmText widget. For best results, define all three target types. When text is imported from an

XmText/XmTextField widget, the string specified by XmNxrtTblImportDelimiter indicates the end of one cell and the start of the next.

converter defines the conversion method to use when dropping a particular type of data; when NULL, the default conversion method for the data type is used. Converters are never defined for XRTTBL_TGT_RANGE, XRTTBL_TGT_COLUMNS, XRTTBL_TGT_ROWS, XRTTBL_TGT_ROWLABELS or XRTTBL_TGT_COLUMNLABELS.

The default value for XmNxrtTblImportTargets is the following list of targets:

```
XRTTBL_TGT_ROWS
XRTTBL_TGT_COLUMNS
XRTTBL_TGT_RANGE
XRTTBL_TGT_INTERNALMOVE
XRTTBL_TGT_CELLS
XRTTBL_TGT_COMPOUND_TEXT
XRTTBL_TGT_TEXT
XRTTBL_TGT_STRING
```

XmNxrtTblIntPickListContext **long *** **NULL** **CSG**

Specifies a list of suggested values for the current context. The value of the first element must be the remaining number of elements in the list. The table makes an internal copy of the values passed.

XmNxrtTblIntPickListSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the pick list as a series of XrtTblPickListValue structures.

XmNxrtTblJumpScroll **XrtTblJumpScroll** **XRTTBL_JUMP_VERTICAL** **CSG**

Specifies the scrolling behavior of each scrollbar. Scrollbars can either scroll smoothly or "jump" scroll in whole row/column increments. When set to XRTTBL_JUMP_NONE, both scrollbars scroll smoothly. When set to XRTTBL_JUMP_HORIZONTAL, the horizontal scrollbar jump scrolls. When set to XRTTBL_JUMP_VERTICAL, the vertical scrollbar jump scrolls. When set to XRTTBL_JUMP_ALL, both scrollbars jump scroll.

XmNxrtTblLabelContext **XmString** **NULL** **CSG**

Specifies a label value, for the current context. Though its data type is XmString, you can specify a String (if linking to Motif 1.2); XRT/table converts it internally to an XmString. Strings can also be specified if XmNxrtTblXmStringSetMode is set appropriately.

When setting this resource, the cell value is copied into each label in the context. When getting this resource, the current context must be a single row or column label.

XmNxrtTblLabelCreateCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called whenever the table displays an empty label (a label with no value supplied for its `XrtTblXmStringList` structure). The callbacks will be called during initial widget display, scrolling and widget exposure.

Each callback is passed a pointer to an `XrtTblLabelCreateCallbackStruct`:

```
typedef struct {
    XrtTblReason        reason;                          // Read-only
    XEvent              *event;                          // Read-only
    int                 row, column;                     // Read-only
    XmString            string;
    Boolean             store;                      // initially False
    Boolean             free_string;                // initially False
XrtTbllabelCreateCallbackStruct;
```

reason is set to `XRTTBL_REASON_LABELCREATE`. *row* and *column* specify the location of the cell. *string* is the cell value returned by your callback routine. *store* should be set to `True` if the string is to be stored in the table; in this case, the callback will not be called again for the label. If *free_string* is `True`, *string* is freed after being displayed.

By default, *string* accepts values of both XmString and String types (unless you are linking to Motif 2.x). `XmNxrtTblXmStringSetMode` can be used to restrict *string* to values of either type String or type XmString.

If this resource is set, the `XmNxrtTblLabelCreateFunc` method is ignored.

XmNxrtTblLabelCreateFunc **XrtTblXmStringProc** **NULL** **CSG**

Specifies a method to be called when displaying a label that is blank in its `XrtTblXmStringList` structure. The method is called during initial widget display, scrolling, and widget exposure. It has the following prototype:

```
XmString label_value_func(Widget table, int row, int column)
```

table is an XRT/table widget; either *row* or *column* must be set to `XRTTBL_LABEL` to specify a row or column label. The value returned may be either a String or an XmString, depending on the value of `XmNxrtTblXmStringSetMode`. The string returned by the method will be freed by the widget after display. *XrtTblCreateSpreadsheetLabel()* may be used to create spreadsheet-style labels.

Use *XtSetValues()* to register this method, not *XtAddCallback()*.

The method is ignored if `XmNxrtTblLabelCreateCallback` has been set.

XmNxrtTblLeftColumn **int** ***dynamic*** **CSG**

Specifies the non-frozen column at least partially visible at the left side of the window. This resource is updated as a user scrolls through a table. When set, the table scrolls to display the column at the left side of the window. This resource is always larger or equal to the number of frozen columns.

XmNxrtTblMarginHeight **Dimension** **2** **CSG**

Specifies the margin (in pixels) between the inside edge of the cell border and the top and bottom edge of the cell's contents. This resource affects the entire table.

XmNxrtTblMarginHeight cannot be smaller than the value of XmText's XmNhighlightThickness resource.

This resource can be set to 0 only when no cell is traversable.

XmNxrtTblMarginWidth **Dimension** **5** **CSG**

Specifies the margin (in pixels) between the inside edge of the cell border and the left/right edge of the cell's contents. This resource affects the entire table. XmNxrtTblMarginWidth cannot be smaller than the value of XmText'sXmNhighlightThickness resource.

This resource can be set to 0 only when no cell is traversable.

XmNxrtTblMaxLengthContext **int** **XRTTBL_MAXINT** **CSG**

Specifies the maximum number of characters that can be entered in a cell by a user, for the current context. This corresponds to the XmNmaxLength resource of the XmText widget. This resource does not affect cell values changed with *XrtTblSetCellValues()*, *XrtTblGetCellValues()* or the macros. Newline characters in multi-line cells count as one character.

XmNxrtTblMaxLengthSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the maximum number of characters that can be entered in a cell by a user, as a series of XrtTblIntValue structures.

XmNxrtTblMinCellVisibility **int** **100** **CSG**

Specifies the minimum amount of a cell that is made visible when traversed to. When the table scrolls to traverse to a non-visible cell, this resource determines the percentage of the cell scrolled into view. When set to 100, the cell is made completely visible. When set to 10, for example, only 10% of the cell is made visible. If set to 0, the cell will not scroll to be made visible. This resource affects the behavior of *XrtTblMakeCellVisible()*.

XmNxrtTblMinFlexibleHeight **Dimension** **5** **CSG**

The minimum height for a row whose XmNxrtTblPixelHeightContext resource is set to XRTTBL_FLEXIBLE.

XmNxrtTblMinFlexibleWidth **Dimension** **5** **CSG**

The minimum width for a row whose XmNxrtTblPixelWidthContext resource is set to XRTTBL_FLEXIBLE.

XmNxrtTblMode **XrtTblMode** **XRTTBL_MODE_TABLE** **CSG**

Specifies the display mode, which must be one of:

XRTTBL_MODE_LIST	XmList operation
XRTTBL_MODE_TABLE	table operates normally

When this resource is set to XRTTBL_MODE_LIST, selecting a cell causes all cells in the row to be selected.

The XmNxrtTblSelectionPolicy resource controls whether users are allowed to select single rows, a range of rows, or multiple ranges of rows.

XmNxrtTblMultilineContext **Boolean** **False** **CSG**

Specifies whether a user can enter multiple lines of text into a cell (that is, whether the XmText widget's `XmNeditMode` resource is set to `XmMULTI_LINE_EDIT`), for the current context. This resource does not affect cell values added or changed by the application. If the cell's value is already multiple lines, a multiple line XmText widget is displayed when the cell is traversed to, regardless of the `XmNxrtTblMultilineContext` value.

XmNxrtTblMultilineSeries **XrtTblSeries *** **NULL** **CSG**

Specifies multi-line capability of cells as a series of `XrtTblBooleanValue` structures.

XmNxrtTblNumColumns **int** **5** **CSG**

Specifies the current number of columns in the table. This resource does not affect the table's `XrtTblCellValues` structure. *XrtTblAddColumns()* and *XrtTblDeleteColumns()* update this resource by the number of columns added/deleted. If set to zero, no cells are displayed; however, any row labels defined are shown.

XmNxrtTblNumRows **int** **10** **CSG**

Specifies the current number of rows in the table. This resource does not affect the table's `XrtTblCellValues` structure. *XrtTblAddRows()* and *XrtTblDeleteRows()* update this resource by the number of rows added/deleted. If set to zero, no cells are displayed; however, any column labels defined are shown.

XmNxrtTblOverflowContext **XrtTblOverflow** **XRTTBL_OVERFLOW_NONE** **CSG**

Specifies whether cell or label values can overflow their cell boundaries, for the current context. When set to `XRTTBL_OVERFLOW_NONE`, text is clipped within the cell boundaries. When set to `XRTTBL_OVERFLOW_HORIZONTAL`, text can flow over blank horizontally-adjacent cells.

Overflow direction depends on the cell's horizontal text alignment (specified by `XmNxrtTblAlignmentContext`). Cell/label values set by a callback or method(`XmNxrtTblCellValueCallback`, `XmNxrtTblCellValueFunc`, `XmNxrtTblLabelCreateCallback` and `XmNxrtTblLabelCreateFunc`) cannot overflow. Row labels cannot overflow. A spanned cell cannot overflow.

Do not set this resource if you have set `XmNxrtTblWordWrapContext` or `XmNxrtTblWordWrapSeries`.

XmNxrtTblOverflowSeries **XrtTblSeries *** **NULL** **CSG**

Specifies whether cell values can overflow their cell boundaries, as a series of `XrtTblIntValue` structures.

Do not set this resource if you have set `XmNxrtTblWordWrapContext` or `XmNxrtTblWordWrapSeries`.

XmNxrtTblPixelHeightContext **int** **XRTTBL_NOVALUE** **CSG**

Specifies row height in pixels, for the current context. Cell values, pixmaps, or widgets that do not fit vertically in a cell are clipped. This resource controls the height unless it is set to

XRTTBL_NOVALUE (the default)—XmNxrtTblCharHeightContext then controls the height. When setting this resource, the context should be an entire row (*x*, XRTTBL_ALL).

The following special values are valid:

XRTTBL_VARIABLE	set to the highest value in the row
XRTTBL_AS_IS	don't change
XRTTBL_FLEXIBLE	shrink or expand depending on table height

A flexible row changes size to ensure that the sum of all flexible rows plus the sum of all other rows is equal to the table height. All flexible rows are set to the same height. If no space is available, the height of all flexible rows is set to the value of the XmNxrtTblMinFlexibleHeight resource.

When a user interactively resizes a row, this resource updates. If character specification was in effect for the row, it is overridden by the pixel height of the user-resized row.

XmNxrtTblPixelHeightSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the height of rows in pixels as a series of XrtTblIntValue structures.

XmNxrtTblPixelWidthContext **int** **XRTTBL_NOVALUE** **CSG**

Specifies column width in pixels, for the current context. Cell values, pixmaps, or widgets that do not fit horizontally in a cell are clipped. This resource controls the width unless it is set to XRTTBL_NOVALUE (the default)—XmNxrtTblCharWidthContext then controls the width. When setting this resource, the context should be an entire column (XRTTBL_ALL, *x*).

The following special values are valid:

XRTTBL_VARIABLE	set to the highest value in the column
XRTTBL_AS_IS	don't change
XRTTBL_FLEXIBLE	shrink or expand depending on table width

A flexible column changes size to ensure that the sum of all flexible columns plus the sum of all other columns is equal to the table width. All flexible columns are set to the same width. If no space is available, the width of all flexible columns is set to the value of the XmNxrtTblMinFlexibleWidth resource.

When a user interactively resizes a column, this resource updates. If character specification was in effect for the column, it is overridden by the pixel width of the user-resized column.

XmNxrtTblPixelWidthSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the width of columns in pixels as a series of XrtTblIntValue structures.

XmNxrtTblPixmapContext **Pixmap** **XmUNSPECIFIED_PIXMAP** **CSG**

Specifies the pixmap displayed, for the current context. When a cell also contains a value, the pixmap is displayed in the background, behind the text. When a cell contains a widget, XRT/table sets the widget's XmNbackgroundPixmap resource to this pixmap.

XmNxrtTblPixmapLayoutContext determines the positioning of the pixmap within the cell. The pixmap is not copied.

XRT/table's resource converters support pixmaps in XBM and XPM format.

XmNxrtTblPixmapLayoutContext **XrtTblPixmapLayout** **XRTTBL_PIXMAP_MIDDLECENTER** **CSG**

Specifies the layout of the background pixmap (set by XmNxrtTblPixmapContext), for the current context. Valid values are XRTTBL_PIXMAP_TILE and nine values that position the pixmap within the cell, shown below:

XRTTBL_PIXMAP_TOPLEFT	XRTTBL_PIXMAP_TOPCENTER	XRTTBL_PIXMAP_TOPRIGHT
XRTTBL_PIXMAP_MIDDLELEFT	XRTTBL_PIXMAP_MIDDLECENTER	XRTTBL_PIXMAP_MIDDLERIGHT
XRTTBL_PIXMAP_BOTTOMLEFT	XRTTBL_PIXMAP_BOTTOMCENTER	XRTTBL_PIXMAP_BOTTOMRIGHT

When set to XRTTBL_PIXMAP_TILE, XRT/table uses the pixmap as a tile pattern.

XmNxrtTblPixmapLayoutSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the layout of background pixmaps as a series of XrtTblIntValue structures.

XmNxrtTblPixmapSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the background pixmap as a series of XrtTblIntValue structures.

XmNxrtTblPixmapShadowContext **Boolean** **True** **CSG**

Specifies whether a cell or label's shadow should be drawn if it contains a pixmap. When set to False, the pixmap is drawn to the cell's edge.

XmNxrtTblPixmapShadowSeries **XrtTblSeries *** **NULL** **CSG**

Specifies, as a series of XrtTblBooleanValue structures, whether the shadows of cells/labels which contain pixmaps should be drawn.

XmNxrtTblPreviewScrolling **XrtTblPreviewScroll** **XRTTBL_PREVIEW_NONE** **CSG**

Specifies whether or not to display a small window next to the scrollbar thumb when the scrollbar thumb is dragged, showing the row/column that would be displayed at the top-left corner of the table if the thumb were released. The default value is XRTTBL_PREVIEW_NONE, meaning the window does not display. All the available values are listed below:

```
XRTTBL_PREVIEW_NONE
XRTTBL_PREVIEW_VERTICAL
XRTTBL_PREVIEW_HORIZONTAL
XRTTBL_PREVIEW_BOTH
```

If the above value is not XRTTBL_PREVIEW_NONE, the table does not redraw until the scrolling is complete (that is, the scroll thumb is released).

The window's appearance is controlled by many of the same attributes which control the table's appearance, itself. The foreground color is XRT/table's XmNforeground. The background color is XRT/table's XmNbackground. The fontlist is XRT/table's default fontlist.

This is either the first entry in the XmNxrtTblFontListSeries with the context (ALL, ALL), or "fixed" if no such entry exists. The default border is XRTTBL_BORDER_NONE.

These values can be adjusted by the user with the following functions:

```
void
XrtTblSetPreviewProperties(
    Widget          w,
    Pixel           foreground,
    Pixel           background,
    XmFontList      fontlist,  /* This value is copied internally */
    XrtTblBorderTypeborder
);

void
XrtTblGetPreviewProperties(
    Widget          w,
    Pixel           *foreground,/* Returned */
    Pixel           *background,/* Returned */
    XmFontList      *fontlist,  /* Returned. This value is not copied
                                   and should not be freed. */
    XrtTblBorderType*border     /* Returned */
);
```

XmNxrtTblPrintCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called as each page of a table is printed. Each routine is passed a pointer to an XrtTblPrintCallbackStruct:

```
typedef struct {
    int             reason;         // Read-only
    XEvent          *event;         // Read-only
    XrtTblRange     **range;        // Read-only
    int             current_page;   // Read-only
    int             total_pages;    // Read-only
    int             across_pages;   // Read-only
    int             down_pages;     // Read-only
    int             page_number;
    String          header, footer;
    FILE            *fp;
    Boolean         write_file_header;
    Boolean         write_file_trailer;
    Boolean         doit;           // initially True
} XrtTblPrintCallbackStruct;
```

reason is either XRTTBL_REASON_PRINT_BEGIN or XRTTBL_REASON_PRINT_END. *event* is always NULL. *range* is the range of cells on the page. *current_page* is the current page being printed. *across_pages* is the number of pages needed to span the width of the table. *down_pages* is the number of pages needed to span the height of the table. *total_pages* is the total number of pages to be printed (equal to *across_pages* × *down_pages*). *page_number* is the current printed page number, which is printed whenever the # symbol is specified for the page header or footer (by *XrtTblVaDrawPS()*). *header* and *footer* are the current page header and footer. *fp* is the current output file pointer. *write_file_header* indicates whether the PostScript header is to be written out. (*write_file_header* is True when *current_page* is 1.) *write_file_trailer* indicates whether the PostScript trailer is to be written out; it is automatically set to True when *current_page* =

total_pages. *doit* indicates whether to skip this page when printing; it has no effect on *current_page* or *total_page*.

If *header* or *footer* is changed, you must free the existing header or footer and allocate a new String.

Pages are printed in left-to-right, then top-to-bottom order.

XmNxrtTblPrintWidgetCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called as each widget in a cell in a table is being printed. Each routine is passed a pointer to an `XrtTblPrintWidgetCallbackStruct`:

```
typedef struct {
    int             reason;         // Read-only
    XEvent          *event;         // Read-only
    Widget          widget;         // Read-only
    int             row, column;    // Read-only
    FILE            *fp;            // Read-only
    int             current_page;   // Read-only
    int             page_number;    // Read-only
    Boolean         doit;           // initially True
} XrtTblPrintWidgetCallbackStruct;
```

reason is either `XRTTBL_REASON_PRINTWIDGET_BEGIN` or `XRTTBL_REASON_PRINTWIDGET_END`. *event* is always `NULL`. *row* and *column* specify the cell being printed; if a title widget is being printed, *row* and *column* have the value `XRTTBL_NOVALUE`. *fp* is the current output file pointer. *current_page* is the number of pages printed thus far. *page_number* is the current page number, which is printed whenever the # symbol is specified for the page header or footer (by *XrtTblVaDrawPS()*). *doit* may be set to `False` if the widget is not to be printed.

XmNxrtTblPSFontMapList **XrtTblPSFontMap *** **NULL** **CSG**

Defines a list of PostScript font to X font correspondences, with an optional point size. Each element in the list must be of type `XrtTblPSFontMap`:

```
typedef struct {
    String          font_name, tag;
    String          PS_font;
    double          point_size;                                  // 0 = infer
} XrtTblPSFontMap;
```

The X font can be represented by either its full font name (*font_name*) or its `XmNfontList` tag (*tag*). *PS_font* is the PostScript font, and *point_size* is the point size; if *point_size* is 0, it is inferred from the font. The last item in the list must have the *PS_font* member set to `NULL`.

XmNxrtTblRangeContext **XrtTblRange *** **NULL** **CSG**

Specifies a range as the current context.[1] When set to a valid range, this resource defines the current context, ignoring the value of `XmNxrtTblContext`. Subsequent Context resource settings are applied to each cell or label in the range. If `XmNxrtTblContext` is set, this resource is set to `NULL` as a side-effect.

Use *XrtTblSetRange()* to set this resource. `XmNxrtTblRangeContext` cannot be set in a resource file.

1. See section 2.4 on page 25 for more information on *contexts* and Context resources.

XmNxrtTblRepaint **XrtTblRepaint** **XRTTBL_REPAINT_ON** **CSG**

Specifies whether the XRT/table widget should be redrawn and recomputed whenever one of its resources is set. Valid values are:

XRTTBL_REPAINT_ON	draw as needed, cell and table sizes kept up to date
XRTTBL_REPAINT_OFF	no drawing, but cell and table sizes kept up to date
XRTTBL_REPAINT_NOCALC	no drawing or size calculation performed

XmNxrtTblRepeatBackgroundColors **String *** **NULL** **CSG**

A list of colors to be used when an XmNxrtTblBackgroundSeries value is XRTTBL_REPEAT_ROW or XRTTBL_REPEAT_COLUMN. The last element in the list must be NULL.

XmNxrtTblRepeatForegroundColors **String *** **NULL** **CSG**

A list of colors to be used when an XmNxrtTblForegroundSeries value is XRTTBL_REPEAT_ROW or XRTTBL_REPEAT_COLUMN. The last element in the list must be NULL.

XmNxrtTblResizeCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application before and after resizing the widget. A pointer to an XrtTblResizeCallbackStruct is passed to each callback. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget              table,
    XtPointer           client_data,
    XtPointer           call_data   // points to an XrtTblResizeCallbackStruct
        )
```

where XrtTblResizeCallbackStruct is defined as:

```
    typedef struct {
        XrtTblReason        reason;                 // Read-only
        XEvent              *event;                 // Read-only
        Dimension           width, height;          // Read-only
        Dimension           visible_width;          // Read-only
        Dimension           visible_height;         // Read-only
    } XrtTblResizeCallbackStruct;
```

reason is either XRTTBL_REASON_RESIZE_BEGIN or XRTTBL_REASON_RESIZE_END; *width* and *height* specify the current or new widget dimensions; *visible_width* and *visible_height* specify the dimensions of the visible cell/label area of the widget.

XmNxrtTblResizeCellCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called as a user resizes a row or column. ResizeCell callbacks are triggered by the *ResizeCell()* action. A pointer to an

XrtTblResizeCellCallbackStruct is passed to each callback. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget          table,
    XtPointer       client_data,
    XtPointer       call_data   // points to an XrtTblResizeCellCallbackStruct
        )
```

where XrtTblResizeCellCallbackStruct is defined as:

```
    typedef struct {
        XrtTblReason    reason;                         // Read-only
        XEvent          *event;                         // Read-only
        String          *params;                        // Read-only
        Cardinal        num_params;                     // Read-only
        int             row, column;                    // Read-only
        int             current_row_height,             // Read-only
        int             current_column_width,           // Read-only
        int             new_row_height,
        int             new_column_width,
        Boolean         doit;
        } XrtTblResizeCellCallbackStruct;
```

reason is either XRTTBL_REASON_RESIZE_BEGIN, XRTTBL_REASON_RESIZE_END or XRTTBL_REASON_RESIZE_DONE; *row, column* specify the row and/or column being resized (if resizing only a row, column is XRTTBL_NOVALUE and vice versa); *current_row_height* and *current_column_width* are the current height/width of the row/column; *new_row_height* and *new_column_width* are the new height/width of the row/column (set by the user's resize); *doit* is a flag that determines whether to allow the interactive resize.

XmNxrtTblRowLabelOffset **int** **0** **CSG**

Specifies the spacing between the row labels and the cell area in pixels. If a negative value is specified, the labels will overlap the cell area.

XmNxrtTblRowLabelPlacement **XrtTblColumnPlacement** **XRTTBL_PLACE_LEFT** **CSG**

Specifies the position of the row labels on the screen. Valid values are XRTTBL_PLACE_LEFT and XRTTBL_PLACE_RIGHT.

XmNxrtTblRowLabels **XrtTblXmStringList *** **NULL** **CSG**

Specifies all of the row labels. Though its data type is XmString, you can specify a String (if linking to Motif 1.2); XRT/table converts it internally to an XmString. Strings can also be specified if XmNxrtTblXmStringSetMode is set appropriately.

XmNxrtTblScrollCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called when the table is scrolled. Each routine is passed a pointer to an `XrtTblScrollCallbackStruct`:

```
typedef struct {
    int                     reason;                          // Read-only
    XEvent                  *event;                          // Read-only
    unsigned char           direction;                       // Read-only
    int                     new_value;
    String                  preview_text                    // Read/Write
    XmScrollBarCallbackStruct *sb_cbs;                       // Read-only
} XrtTblScrollCallbackStruct;
```

reason is either `XRTTBL_REASON_SCROLL_BEGIN` or `XRTTBL_REASON_SCROLL_END`; *event* is the passed XEvent; *direction* is either `XmHORIZONTAL` or `XmVERTICAL`; *new_value* is the new value of the `XmNxrtTblTopRow` or `XmNxrtTblLeftColumn` resource; *preview_text* is the text that would be displayed in the preview window. To replace the value, first `XtFree()` the old value and allocate memory for the new value. XRT/table will free the memory used by this element of the callback structure. *sb_cbs* is a pointer to the callback structure passed to the ScrollBar callback.

XmNxrtTblSelectCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called as a user selects cells. Select callbacks are triggered by the *Select()* action. The callbacks are called for every cell selected in a range. A pointer to an `XrtTblSelectCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget          table,
    XtPointer       client_data,
    XtPointer       call_data       // points to an XrtTblSelectCallbackStruct
    )
```

where `XrtTblSelectCallbackStruct` is defined as:

```
typedef struct {
    XrtTblReason    reason;         // Read-only
    XEvent          *event;         // Read-only
    String          *params;        // Read-only
    Cardinal        num_params;     // Read-only
    int             range_index;    // Read-only
    XrtTblRange     *range;         // Read-only
    int             row, column;
    int             click_count;
    Boolean         doit;
} XrtTblSelectCallbackStruct;
```

reason is either `XRTTBL_REASON_SELECT_BEGIN`, `XRTTBL_REASON_SELECT_END`, `XRTTBL_REASON_SELECT_EXTEND`, or `XRTTBL_REASON_UNSELECT`; *range_index* is the position of *range* in the selected cell list (specified by `XmNxrtTblSelectedCellList`); *range* is the range of cells that have been selected (or will be unselected); *row, column* specify the location of the cell or label being selected or unselected; *click_count* specifies the number of user mouse clicks within the multi-click delay (specified by `XmNmultiClickTime`); *doit* is a flag that determines whether the selection/unselection should be allowed.

XmNxrtTblSelectedBackground **String** **"Black"** **CSG**

Specifies the background color of selected cells. The value can be any color from the X color database or a hexadecimal RGB color value. When set to "None", selected cells look identical to unselected cells–the background *and* foreground colors are the same as the colors defined for the cells (`XmNxrtTblForegroundContext` and `XmNxrtTblBackgroundContext`).

On a monochrome Server, setting this resource to anything other than black causes white to be used. See section 2.8 on page 29 for details of color specification.

XmNxrtTblSelectedCellList **XrtTblRange **** **NULL** **CSG**

Specifies a list of currently-selected ranges of cells; the last element of the list must be `NULL`. This resource is updated dynamically when a user selects cells. The selection policy (specified by `XmNxrtTblSelectionPolicy`) controls the amount of selection allowed on the table, both by users and by the application. Each selected range is defined by the following structure:

```
typedef struct {
    int     start_row, start_column;
    int     end_row, end_column;
} XrtTblRange;
```

The start and end locations in each range define the opposite corners of the range. Users can select in any direction, so *start_row* and/or *start_column* may be greater than *end_row* and/or *end_column*. When a user clicks a row/column label to select an entire row or column, *end_row* or *end_column* are set to `XRTTBL_MAXINT`.

Use *XrtTblGetSelectedRange()* to conveniently retrieve one range from the selected range list.

XmNxrtTblSelectedForeground **String** **"White"** **CSG**

Specifies the foreground color of selected cells. The value can be any color from the X color database or a hexadecimal RGB color value. If set to "None" when `XmNxrtTblSelectedBackground` is not "None", the foreground of selected cells is the same as the foreground colors defined for the cells (`XmNxrtTblForegroundContext`). This resource is ignored when `XmNxrtTblSelectedBackground`; is set to "None".

On a monochrome Server, setting this resource to anything other than white causes black to be used. See section 2.8 on page 29 for details of color specification.

XmNxrtTblSelectionPolicy **XrtTblSelectionPolicy** **XRTTBL_SELECT_NONE** **CSG**

Specifies the selection behavior allowed on the table, both by end-users, and by the application.
Valid values are:

XRTTBL_SELECT_NONE	No selection is allowed
XRTTBL_SELECT_SINGLE	Selection of only one cell/row allowed
XRTTBL_SELECT_RANGE	Selection of only one range of cells/rows allowed
XRTTBL_SELECT_MULTIRANGE	Selection of multiple ranges of cells/rows allowed

The XmNxrtTblMode resource controls whether cells or rows are selected.

When XmNxrtTblSelectionPolicy is set to XRTTBL_SELECT_NONE, Select callbacks (specified with XmNxrtTblSelectCallback) are not called as a user traverses or attempts to select cells. Setting this resource does not change the selected cell list (XmNxrtTblSelectedCellList).

XmNxrtTblSensitiveContext **Boolean** **True** **CSG**

Specifies whether a cell is sensitive or insensitive for the current context. An insensitive cell is grayed out, but the user can still select it and traverse into it.

Note: If the entire table is insensitive, this resource has no effect.

XmNxrtTblSensitiveSeries **XrtTblSeries *** **NULL** **CSG**

Specifies cell sensitivity as a series of XrtTblBooleanValue structures.

This resource is available in ContextValueCallback. See "Managing Contexts Programmatically" on page 139.

XmNxrtTblSetTextCursorPos **Boolean** **False** **CSG**

Specifies whether the insertion cursor in the XmText widget should be initially set to the mouse click position inside the cell.

Do not set this resource if you have set the XmNxrtTblTextInitSelect resource.

XmNxrtTblShadowThickness **Dimension** **2** **CSG**

Specifies the width of the borders that surround all cells and labels.

XmNxrtTblSortCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called after a call to *XrtTblSortTable()*. Each routine is passed a pointer to an `XrtTblSortCallbackStruct`:

```
typedef struct {
    int                 reason;     // Read-only
    XEvent              *event;     // Read-only
    int                 column;     // Read-only
    XrtTblCompareStruct *data;      // Read-only
    int                 num_rows;   // Read-only
} XrtTblSortCallbackStruct;
```

reason is always `XRTTBL_REASON_SORT`; *event* is `NULL`; *column* is the column which has been sorted; *data* is the new sorted order (`data[i].row` is moved to row `i`); *num_rows* is the length of the data.

XmNxrtTblSpace **Dimension** **6** **CSG**

Specifies the number of pixels between the scrollbars and the table's cells/labels. This value is not affected by the scrollbar's attachment (specified by `XmNxrtTblHorizScrollBarPosition` and `XmNxrtTblVertScrollBarPosition`).

XmNxrtTblSpanList **XrtTblRange **** **NULL** **CSG**

Specifies a list of currently-spanned ranges of cells or labels; the last element of the list must be `NULL`. A spanned range is a range of cells or labels that appear joined and can be treated as one cell. Each spanned range is defined by the following structure:

```
typedef struct {
    int     start_row, start_column;
    int     end_row, end_column;
} XrtTblRange;
```

The top-left cell (specified by the *start_row* and *start_column* members) is the source cell for the spanned range. The cell/label value and attributes of the source cell are displayed in the spanned cell. Attributes for the spanned range must be set on the source cell.

A spanned range cannot include both cells and labels. There must also be more than one cell/label in a spanned range; when a single-cell range is specified, XRT/table removes it from the list.

XmNxrtTblStringPickListContext **String *** **NULL** **CSG**

Specifies a list of suggested values for the current context. The list of values must be `NULL` terminated. The table makes an internal copy of the values passed.

XmNxrtTblStringPickListSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the pick list as a series of `XrtTblPickListValue` structures.

XmNxrtTblText **Widget** **NULL** **SG**

Specifies the widget ID of the XmText widget used for cell editing.

XmNxrtTblTextInitSelect **Boolean** **False** **SG**

Specifies whether the text in the current cell should be initially selected upon traversal. When set to `True`, *XmTextSetSelection()* is called to select the text.

Do not set this resource if you have set the `XmNxrtTblSetTextCursorPos` resource.

XmNxrtTblTopRow **int** ***dynamic*** **CSG**

Specifies the non-frozen row at least partially visible at the top of the window. This resource is updated as a user scrolls through a table. When set, the table scrolls to display the row at the top of the window. This resource is always greater than or equal to the number of frozen rows.

XmNxrtTblTopShadowContext **Pixel** **XRTTBL_NOVALUE** **CSG**

Specifies the top shadow color for the current context. When set to `XRTTBL_NOVALUE`, XRT/table automatically calculates appropriate colors from `XmNxrtTblBackgroundContext`. This resource must be set to a Motif Pixel color value.

This resource has no corresponding Series resource – XRT/table stores shadow colors in the `XmNxrtTblBackgroundSeries`.

XmNxrtTblTrackCursor **Boolean** **True** **CSG**

Specifies whether the mouse pointer should be tracked as a user moves the mouse across the widget. By default, XRT/table tracks the mouse pointer and changes its shape to reflect the interactions available to the user at that location.

XmNxrtTblTrackCursorCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called as a user moves the mouse pointer across the table. TrackCursor callbacks are triggered as the pointer crosses cell/label borders. An application can use this callback to change the cursor shape at particular areas of the table. A pointer to an `XrtTblTrackCursorCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
    CallbackProcedure(
    Widget          table,
    XtPointer       client_data,
    XtPointer       call_data   // points to an XrtTblTrackCursorCallbackStruct
    )
```

where `XrtTblTrackCursorCallbackStruct` is defined as:

```
    typedef struct {
        XrtTblReason            reason;             // Read-only
        XEvent                  *event;             // Read-only
        int                     row, column;        // Read-only
        XrtTblBorderSides       sides;              // Read-only
        Boolean                 enter_cell;         // Read-only
        Cursor                  old_cursor;         // Read-only
        Cursor                  new_cursor;
    } XrtTblTrackCursorCallbackStruct;
```

reason is `XRTTBL_REASON_CURSOR_LEAVE` when leaving a cell/label, `XRTTBL_REASON_CURSOR_EDGE` when at the edge of a cell/label boundary, or `XRTTBL_REASON_CURSOR_INSIDE` when inside a cell/label[1]; *row, column* specify the location of the mouse pointer (`XRTTBL_NOVALUE` when outside of cells/labels); *sides* specifies the border

side that the mouse pointer is over; *enter_cell* is `True` when the pointer is in a cell not previously reported; *old_cursor* and *new_cursor* are the current and new mouse pointers. A callback routine can change *new_cursor* to *old_cursor* to disallow a cursor change.

XmNxrtTblTraversableContext **Boolean** **True** **CSG**

Specifies whether a cell can be traversed to by a user or an application, for the current context. A cell which is not traversable cannot be edited by a user.

XmNxrtTblTraversableSeries **XrtTblSeries *** **NULL** **CSG**

Specifies cell traversability as a series of `XrtTblBooleanValue` structures.

XmNxrtTblTraverseCellCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called as a user traverses from one cell to another. TraverseCell callbacks are triggered by the *TraverseToCell()* action, and when the table gets focus while the XmText widget is unmapped. This callback is called after a user's cell edit has been committed, before moving the XmText widget to the next cell. A pointer to an `XrtTblTraverseCellCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
    CallbackProcedure(
    Widget          table,
    XtPointer       client_data,
    XtPointer       call_data  // points to an XrtTblTraverseCellCallbackStruct
    )
```

where `XrtTblTraverseCellCallbackStruct` is defined as:

```
    typedef struct {
        XrtTblReason    reason;         // Read-only
        XEvent          *event;         // Read-only
        String          *params;        // Read-only
        Cardinal        num_params;     // Read-only
        int             row, column;    // Read-only
        int             next_row;
        int             next_column;
    } XrtTblTraverseCellCallbackStruct;
```

reason is always `XRTTBL_REASON_TRAVERSE_CELL`; *row, column* specify the location of the current cell; *next_row, next_column* are the proposed location of the cell to traverse to.

XmNxrtTblUserDataContext **XtPointer** **NULL** **CSG**

Specifies a user data pointer that the application can attach, for the current context. This resource is not used by XRT/table.

XmNxrtTblUserDataSeries **XrtTblSeries *** **NULL** **CSG**

Specifies user data as a series of `XrtTblPointerValue` structures.

---

1. *reason* is also `XRTTBL_REASON_CURSOR_INSIDE` when the mouse is outside of the cell/label area (for example, over a scrollbar or title)—but in this case, *row* and *column* are `XRTTBL_NOVALUE`.

XmNxrtTblUseXmGetColors **Boolean** **True** **CSG**

Specifies whether *XmGetColors()* should be called to calculate the cell shadow colors. If `False`, use the XRT/table 2.0 algorithm to calculate shadows.

XmNxrtTblValidateCellCallback **XtCallbackList** ***see below*** **CSG**

Specifies a list of callbacks that are called when a cell/label has been edited, before and after committing the edit to cell or label values structures. ValidateCell callbacks are triggered by the *TraverseToCell()* and *CommitEdit()* actions. A pointer to an `XrtTblValidateCellCallbackStruct` is passed to each callback. The default callback is *XrtTblCheckDispatcher()*; this routine validates the edited cell/label value and changes the `XrtTblValidateCellCallbackStruct` structure. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget          table,
    XtPointer       client_data,
    XtPointer       call_data // points to an XrtTblValidateCellCallbackStruct
    )
```

where `XrtTblValidateCellCallbackStruct` is defined as:

```
    typedef struct {
        XrtTblReason        reason;                 // Read-only
        XEvent              *event;                 // Read-only
        int                 row, column;            // Read-only
        Boolean             value_changed;          // Read-only
        XrtTblDataType      datatype;               // Read-only
        XtArgVal            *converted_value;       // Read-only
        XtPointer           user_data;              // Read-only
        String              value;
        Boolean             doit;
        Boolean             bell;
    } XrtTblValidateCellCallbackStruct;
```

reason is `XRTTBL_REASON_VALIDATE_CELL_BEGIN` (before committing the edit) or `XRTTBL_REASON_VALIDATE_CELL_END` (after committing the edit); *row*, *column* specify the location of the current cell; *value_changed* is `True` if the user has attempted to change the cell value; *datatype* is the `XmNxrtTblDataTypeContext` value for the cell; *converted_value* is a pointer to the converted value determined by the validation method; *user_data* is the `XmNxrtTblUserDataContext` value for the cell; *value* contains the data to be committed if *doit* is `True`; *doit* should be set to `False` if the value should not be committed; *bell* indicates whether to ring the terminal bell to indicate that the edited cell has not been committed (*bell* is `True` when *doit* is `False`, and vice versa).

The ValidateCell callbacks are only called with the `XRTTBL_REASON_VALIDATE_CELL_END` reason if the value has been changed interactively via a cell edit (which happens if `XmNxrtTblValidatePolicy` is set to `XRTTBL_VALIDATE_USER_EDIT`).

XmNxrtTblValidatePolicy **XrtTblValidatePolicy** **XRTTBL_VALIDATE_USER_EDIT** **CSG**

Specifies when to validate changes to cell or label values against its data type (specified by `XmNxrtTblDataTypeContext`). When set to `XRTTBL_VALIDATE_USER_EDIT`, only changes made by a user are validated. When set to `XRTTBL_VALIDATE_SET`, only changes made by an application are validated. When set to `XRTTBL_VALIDATE_ALWAYS`, changes made by both a user

and an application are validated. When set to XRTTBL_VALIDATE_NEVER, changes are never validated.

XRT/table validates cell/label changes using its own ValidateCell callback routine (specified by XmNxrtTblValidateCellCallback).

XmNxrtTblVerticalScrollBar **Widget** **NULL** **SG**

Specifies the widget ID of the vertical scrollbar. Do not set to NULL if you want to disable display of the scrollbar–use XmNxrtTblDisplayVertScrollBar.

XmNxrtTblVertScrollBarBottomAttachment **XrtTblAttachment** **XRTTBL_ATTACH_CELLS** **CSG**

Specifies how the bottom end of the vertical scrollbar is attached to the table. When set to XRTTBL_ATTACH_CELLS, the scrollbar ends at the bottom edge of the visible cells. When set to XRTTBL_ATTACH_SIDE, the scrollbar ends at the bottom edge of the widget.

XmNxrtTblVertScrollBarPlacement **XrtTblColumnPlacement** **XRTTBL_PLACE_RIGHT** **CSG**

Specifies the side of the table to place the vertical scrollbar. When set to XRTTBL_PLACE_RIGHT, the scrollbar is placed to the right of the table. When set to XRTTBL_PLACE_LEFT, the scrollbar is placed to the left of the table.

XmNxrtTblVertScrollBarPosition **XrtTblSBPosition** **XRTTBL_SBPOSITION_CELLS** **CSG**

Specifies how the vertical scrollbar is attached to the table. When set to XRTTBL_SBPOSITION_CELLS, the scrollbar is attached to the cell/label viewport. When set to XRTTBL_SBPOSITION_SIDE, the scrollbar is attached to the side of the widget. XmNxrtTblSpace specifies the distance between the scrollbar and the table.

XmNxrtTblVertScrollBarTopAttachment **XrtTblAttachment** **XRTTBL_ATTACH_CELLS** **CSG**

Specifies how the top end of the vertical scrollbar is attached to the table. When set to XRTTBL_ATTACH_CELLS, the scrollbar ends at the top edge of the visible cells. When set to XRTTBL_ATTACH_SIDE, the scrollbar ends at the top edge of the widget.

XmNxrtTblVisibleColumns **int** ***dynamic*** **CSG**

Specifies the number of non-frozen columns currently visible or partially visible. This resource is updated when a user interactively resizes columns or the widget. When set, the widget attempts to resize to display the specified number of columns, overriding the resize policy (XmNxrtTblAllowResize). Setting this resource to 0 causes XRT/table to attempt to resize to make all columns visible, and sets XmNxrtTblVisibleColumns equal to XmNxrtTblNumColumns (this happens also when creating the widget if this resource is defaulted).

XmNxrtTblVisibleRows **int** ***dynamic*** **CSG**

Specifies the number of non-frozen rows currently visible or partially visible. This resource is updated when a user interactively resizes rows or the widget. When set, the widget attempts to resize to display the specified number of rows, overriding the resize policy (XmNxrtTblAllowResize). Setting this resource to 0 causes XRT/table to attempt to resize to make all rows visible, and sets XmNxrtTblVisibleRows equal to XmNxrtTblNumRows (this happens also when creating the widget if this resource is defaulted).

XmNxrtTblWidgetClassList **XrtTblWidgetClass *** ***see below*** **CSG**

Specifies a list of widget class pointers for each widget class name; the last element of the list must be `NULL`. This list defines the widget classes that XRT/table recognizes and can create from a resource file. When set, the value is added to the current list. The default list contains all OSF/Motif, XRT/table, and XrtLabel widget classes and class pointers.

XmNxrtTblWidgetContext **Widget** **NULL** **CSG**

Specifies the widget displayed in a cell/label, for the current context. When set, XRT/table creates only the number of copies (clones) needed for the display. XRT/table recognizes all OSF/Motif, XRT/table, XRT/field, and XrtLabel widget classes, and any classes added to `XmNxrtTblWidgetClassList`. When set to an unrecognized widget class, you must use a CreateWidget callback to create the widget manually.

The special context `(XRTTBL_LABEL, XRTTBL_LABEL)` can be used to place a widget at the intersection of the row and column label areas.

If this resource is set to `NULL` for a particular context, any widget previously specified for that context will be unmanaged (but not destroyed).

XmNxrtTblWidgetSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the widget displayed in cells and/or labels as a series of `XrtTblWidgetValue` structures.

XmNxrtTblWidgetSetValueCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called for each cell containing a widget. The routines are called when resources are set on the widget, allowing the application to disable or modify the resource values. Each routine is passed a pointer to an `XrtTblWidgetSetValueCallbackStruct`:

```
typedef struct {
    int         reason;         // Read-only
    XEvent      *event;         // Read-only
    int         row, column;    // Read-only
    Widget      widget;         // Read-only
    Arg         *arglist;       // Read-only
    int         num_arglist;
    Boolean     doit;
} XrtTblWidgetSetValueCallbackStruct;
```

reason is either `XRTTBL_REASON_SETVALUE_BEGIN` or `XRTTBL_REASON_SETVALUE_END`; *event* is always `NULL`; *row*, *column* is the position of the cell or label; *arglist* is the list of args to be set; *doit* is set to `True` if the args are to be applied.

XmNxrtTblWidgetShadowContext **Boolean** **False** **CSG**

Specifies whether a cell or label's shadow should be drawn when it contains a widget, for the current context. If this resource is set to `False`, the widget is sized to the cell's edge.

XmNxrtTblWidgetShadowSeries **XrtTblSeries *** **NULL** **CSG**

Specifies, as a series of `XrtTblBooleanValue` structures, whether the shadows of cells/labels which contain widgets should be drawn.

XmNxrtTblWordWrapContext **Boolean** **False** **CSG**

Specifies whether the text should be displayed as word wrapped (lines broken between words, defined by spaces, tabs or newlines), for the current context. This also sets the XmText `XmNwordWrap` resource when the cell is edited.

Do not set this resource if you have set `XmNxrtTblOverflowContext` or `XmNxrtTblOverflowSeries`.

XmNxrtTblWordWrapSeries **XrtTblSeries *** **NULL** **CSG**

Specifies the word wrap of each cell or label as a series of `XrtTblBooleanValue` structures.

Do not set this resource if you have set `XmNxrtTblOverflowContext` or `XmNxrtTblOverflowSeries`.

XmNxrtTblXmStringGetMode **XrtTblXmStringGet** **XRTTBL_GET_XMSTRING** **CSG**

Specifies how XmString resources are to be retrieved from the table:

`XRTTBL_GET_STRING`	Convert to String
`XRTTBL_GET_XMSTRING`	Get as XmString

In either case, the returned values must not be freed.

XmNxrtTblXmStringSetMode **XrtTblXmStringSet** ***see below*** **CG**

Specifies how XmString resources are set by the application:

`XRTTBL_SET_EXAMINE`	determine at runtime
`XRTTBL_SET_STRING`	Set as String
`XRTTBL_SET_XMSTRING`	Set as XmString

It is illegal to set this resource to `XRTTBL_SET_EXAMINE` when using Motif 2.x. Any attempt to do so will lead to termination of the application.

Under Motif 1.2, the default value for this resource is `XRTTBL_SET_EXAMINE`. Under Motif 2.x, it is `XRTTBL_SET_XMSTRING`.

The value of this resource is ignored when an `XrtTblCellValues` structure is being set directly (since the structure itself specifies the type to use).

If `XmNxrtTblXmStringSetMode` is set to `XRTTBL_SET_STRING`, XmString resources are stored internally as Strings, which is more efficient.

A.3 Constraint Resources

XRT/table defines the following constraint resources for widgets that are children of the table.

XmNxrtTblBottomOffset **int** **XRTTBL_NOVALUE** **CSG**

Specifies the number of pixels between a title's bottom edge and the bottom of the cells/labels (or the bottom of the window when `XmNxrtTblLocation` is `XRTTBL_LOCATION_BOTTOM`). A left- or right-located title is vertically centered when this resource and `XmNxrtTblTopOffset` are `XRTTBL_NOVALUE`.

XmNxrtTblLeftOffset **int** **XRTTBL_NOVALUE** **CSG**

Specifies the number of pixels between a title's left edge and the left edge of the cells/labels (or the left edge of the window when `XmNxrtTblLocation` is `XRTTBL_LOCATION_LEFT`). A top- or bottom-located title is horizontally centered when this resource and `XmNxrtTblRightOffset` are `XRTTBL_NOVALUE`.

XmNxrtTblLocation **XrtTblLocation** **XRTTBL_LOCATION_NONE** **CSG**

Specifies the side of the table to place a title widget. Valid values are: `XRTTBL_LOCATION_LEFT`, `XRTTBL_LOCATION_RIGHT`, `XRTTBL_LOCATION_TOP`, `XRTTBL_LOCATION_BOTTOM` or `XRTTBL_LOCATION_NONE` (not displayed).

If a Manager widget is used as a title, its `XmNxrtTblLocation` resource must be set after its children have been added.

XmNxrtTblResizeToCell **Boolean** **True** **CSG**

Specifies the alignment of a widget in a cell. If set to `False`, then the widget is positioned according to the value of the cell's `XmNxrtTblAlignmentContext`. If it is `True`, then the widget is resized to match the cell size.

XmNxrtTblRightOffset **int** **XRTTBL_NOVALUE** **CSG**

Specifies the number of pixels between a title's right edge and the right edge of the cells/labels (or the right edge of the window when `XmNxrtTblLocation` is `XRTTBL_LOCATION_RIGHT`). A top- or bottom-located title is horizontally centered when this resource and `XmNxrtTblLeftOffset` are `XRTTBL_NOVALUE`.

XmNxrtTblTopOffset **int** **XRTTBL_NOVALUE** **CSG**

Specifies the number of pixels between a title's top edge and the top of the cells/labels (or the top of the window when `XmNxrtTblLocation` is `XRTTBL_LOCATION_TOP`). A left- or right-located title is vertically centered when this resource and `XmNxrtTblBottomOffset` are `XRTTBL_NOVALUE`.

XmNxrtTblWidgetLocation **XrtTblContext *** **NULL** **CSG**

Specifies a widget's location within the table. Any valid context can be used. This resource is equivalent to setting the table's `XmNxrtTblWidgetContext` to the value of the widget.

A.4 Inherited XRT/table Resources

XRT/table inherits resources from its parent classes. The following table alphabetically lists the resources along with the parent class that defines them. Consult your Motif and Xt documentation for more information.

Resource	Inherited From
XmNaccelerators	Core
XmNancestorSensitive	Core
XmNbackground	Core
XmNbackgroundPixmap	Core
XmNborderColor	Core
XmNborderPixmap	Core
XmNborderWidth	Core
XmNbottomShadowColor	XmManager
XmNbottomShadowPixmap	XmManager
XmNchildren	Composite
XmNcolormap	Core
XmNdepth	Core
XmNdestroyCallback	Core
XmNforeground	XmManager
XmNheight	Core
XmNhelpCallback	XmManager
XmNhighlightColor	XmManager
XmNhighlightPixmap	XmManager

Resource	Inherited From
XmNinitialFocus	XmManager
XmNinitialResourcesPersistent	Core
XmNinsertPosition	Composite
XmNmappedWhenManaged	Core
XmNnavigationType	XmManager
XmNnumChildren	Composite
XmNscreen	Core
XmNsensitive	Core
XmNshadowThickness	XmManager
XmNstringDirection	XmManager
XmNtopShadowColor	XmManager
XmNtopShadowPixmap	XmManager
XmNtranslations	Core
XmNtraversalOn	XmManager
XmNuserData	XmManager
XmNwidth	Core
XmNx	Core
XmNy	Core

B

Procedures and Methods Reference

This appendix lists the XRT/table procedures and methods in alphabetical order. Procedures marked "(source)" are also supplied as source code. Source procedures are located in *$XRTHOME/src/table/util*. If you want to change a source procedure, it should be copied to another directory (and linked ahead of XRT/table if its name is not changed).

XmCreateXrtTable()

Creates an XRT/table widget.

```
Widget
XmCreateXrtTable(
    Widget  parent,
    char    *name,
    Arg     *arglist,
    int     argcount
    )
```

parent is the parent; *name* is the created widget's name; *arglist* is a list of *argcount* arguments (created by *XtSetArg()*).

XrtTblAddCallback()

Adds a callback to an XRT/table callback list. This method is equivalent to *XtAddCallback()*, except that it notifies the table that a callback has been added.

```
void
XrtTblAddCallback(
    Widget              table,
    String              cb_name,
    XtCallbackProc      cb,
    XtPointer           tag
)
```

table is an XRT/table widget; *cb_name* is the resource name of the callback list; *cb* is the callback to be added; *tag* is the argument to be passed to the callback procedure.

XrtTblAddCallback() must be used when setting `XmNxrtTblCellValueCallback` and `XmNxrtTblLabelCreateCallback`.

XrtTblAddColumns()

Inserts new columns into the table, shifting any cell values and attributes (such as background color) to the right of the insertion. *XrtTblAddColumns()* returns `False` if any argument is invalid.

```
Boolean
XrtTblAddColumns(
    Widget      table,
    int         position,
    int         num_columns,
    Boolean     shift_labels,
    void        **values,
    int         num_values
    )
```

table is an XRT/table widget; *position* is the column index before which to add the new columns (if set to the value of `XmNxrtTblNumColumns`, columns are added after the last column); *num_columns* is the number of columns to be added; *shift_labels* specifies whether to shift the labels of cells in existing columns to the right of the new columns; *values* is an array of XmStrings or Strings, depending on the value of `XmNxrtTblXmStringSetMode`; *num_values* is the number of elements in *values*. If the current cell is at or below *position*, the edit will be cancelled, and the XmText widget will be unmapped.

XrtTblAddRows()

Inserts new rows into the table, shifting any cell values and attributes (such as background color) below the insertion. *XrtTblAddRows()* returns `False` if any argument is invalid.

```
Boolean
XrtTblAddRows(
    Widget      table,
    int         position,
    int         num_rows,
    Boolean     shift_labels,
    void        **values,
    int         num_values
    )
```

table is an XRT/table widget; *position* is the row before which to add the new rows (if set to the value of `XmNxrtTblNumRows`, rows are added after the last row); *num_rows* is the number of rows to be added; *shift_labels* specifies whether to shift the labels of cells in existing rows below the new rows*; values* is an array of XmStrings or Strings, depending on the value of `XmNxrtTblXmStringSetMode`; *num_values* is the number of elements in *values*. If the current cell is at or below *position*, the edit will be cancelled, and the XmText widget will be unmapped.

XrtTblCancelEdit()

Cancels interactive editing of a cell and optionally unmaps the current cell. Any text entered or changed in the current cell by a user is discarded. *XrtTblCancelEdit()* returns `False` if no edit is in progress or any argument is invalid.

```
Boolean
XrtTblCancelEdit(
    Widget      table,
    Boolean     unmap
    )
```

table is an XRT/table widget; *unmap* specifies whether to unmap the current cell.

XrtTblCellValuesCopy()

Copies an entire cell value structure. The copied structure is returned.

```
XrtTblCellValues *
XrtTblCellValuesCopy(
    XrtTblCellValues *cells
    )
```

XrtTblCheckDispatcher()

Calls a data validation method to validate a cell/label value. This method checks the cell/label's data type (specified by `XmNxrtTblDataTypeContext`) against the list of validation methods (specified by `XmNxrtTblDataTypeList`) and calls the validation method specified for the data type.

```
void
XrtTblCheckDispatcher(
    Widget          table,
    XtPointer       call_data,
    XtPointer       cbs
    )
```

call_data is set when registering the method *XtAddCallback()* (`NULL` by default); *cbs* is a pointer to an `XrtTblValidateCellCallbackStruct` structure.

XrtTblCheckIsBoolean() (source)

Validates a cell/label value as Boolean. This method sets the *doit* member in the passed `XrtTblValidateCellCallbackStruct` structure to `False` if the string is not one of the following: "t", "true", "yes", "on", "1", "f", "false", "no", "off", or "0". String checking is case-insensitive. If the string is empty, *doit* is set to `False`.

```
void
XrtTblCheckIsBoolean(
    Widget          table,
    XtPointer       call_data,
    XtPointer       cbs
    )
```

cbs is a pointer to an `XrtTblValidateCellCallbackStruct` structure.

XrtTblCheckIsDouble() (source)
Validates a cell/label value as a double. This method sets the *doit* member in the passed `XrtTblValidateCellCallbackStruct` structure to `True` if the string is a double or is empty. If the string is empty, the *converted_value* member is set to 0.0.

```
void
XrtTblCheckIsDouble(
    Widget          table,
    XtPointer       call_data,
    XtPointer       cbs
    )
```

cbs is a pointer to an `XrtTblValidateCellCallbackStruct` structure.

XrtTblCheckIsEmpty() (source)
Returns `True` if the string is empty (or only has whitespace characters).

```
Boolean
XrtTblCheckIsEmpty(
    String          string
    )
```

XrtTblCheckIsFloat() (source)
Validates a cell/label value as a float. This method sets the *doit* member in the passed `XrtTblValidateCellCallbackStruct` structure to `True` if the string is a float or is empty. If the string is empty, the *converted_value* member is set to 0.0.

```
void
XrtTblCheckIsFloat(
    Widget          table,
    XtPointer       call_data,
    XtPointer       cbs
    )
```

cbs is a pointer to an `XrtTblValidateCellCallbackStruct` structure.

XrtTblCheckIsInt() (source)
Validates a cell/label value as an integer. This method sets the *doit* member in the passed `XrtTblValidateCellCallbackStruct` structure to `True` if the string is an integer or is empty. If the string is empty, the *converted_value* member is set to 0.

```
void
XrtTblCheckIsInt(
    Widget          table,
    XtPointer       call_data,
    XtPointer       cbs
    )
```

cbs is a pointer to an `XrtTblValidateCellCallbackStruct` structure.

XrtTblCommitEdit()
Commits interactive cell editing to the cell values structure, saving any changes made by the user. The method optionally unmaps the current cell. The callbacks on the `XmNxrtTblValidateCellCallback` list are called first to verify that the changes

are valid. *XrtTblCommitEdit()* returns `False` if no edit is in progress or an argument is invalid.

```
Boolean
XrtTblCommitEdit(
    Widget          table,
    Boolean         unmap
    )
```

table is an XRT/table widget; *unmap* specifies whether to unmap the current cell if the method is successful.

XrtTblCreateSpreadsheetLabel() (source)

For use with the `XmNxrtTblLabelCreateFunc` resource, this method creates spreadsheet-style labels (rows incremented starting at 1, and columns labelled alphabetically starting at A).

```
XmString
XrtTblCreateSpreadsheetLabel(
    Widget      table,
    int         row,
    int         column
    )
```

This method returns a String if `XmNxrtTblXmStringSetMode` is set to `XRTTBL_SET_STRING`. Otherwise it returns an XmString.

XrtTblCvtPixelToString()

Converts a Motif Pixel color value to a hexadecimal RGB color String, for use when setting XRT/table's color resources. *XrtTblCvtPixelToString()* returns `False` if any argument is invalid.

```
Boolean
XrtTblCvtPixelToString(
    Widget      mywidget,
    Pixel       pixel,
    String      string      /* Returned */
    )
```

mywidget is any widget realized in the X display; *pixel* specifies a Pixel color value; *string* is returned containing the converted Pixel color value.

XrtTblCvtStringToXmString()

Converts an String to a XmString. XRT/table's converter is used if *XrtTblSetXmStringConverter()* has been called; otherwise, *XmStringCreateLtoR()* is used.

```
String
XrtTblCvtStringToXmString(
    String          string
    )
```

XrtTblCvtXmStringToString()

Converts an XmString to a String. The String should be freed after use.

```
String
XrtTblCvtXmStringToString(
    XmString        string
    )
```

XrtTblDeleteColumns()

Deletes columns from the table and frees their cell values. *XrtTblDeleteColumns()* returns `False` if any argument is invalid. The attributes (such as background color) of the remaining columns are unaffected by the deletion.

```
Boolean
XrtTblDeleteColumns(
    Widget          table,
    int             position,
    int             num_columns,
    Boolean         shift_labels
    )
```

table is an XRT/table widget; *position* is the first column number to delete; *num_columns* is the number of columns to be deleted; *shift_labels* specifies whether to shift left the labels of columns to the right of the deleted columns. If the current cell is at or to the right of *position*, the edit will be cancelled, and the XmText widget will be unmapped.

XrtTblDeleteRows()

Deletes rows from the table and frees their cell values. *XrtTblDeleteRows()* returns `False` if any argument is invalid. The attributes (such as background color) of the remaining rows are unaffected by the deletion.

```
Boolean
XrtTblDeleteRows(
    Widget          table,
    int             position,
    int             num_rows,
    Boolean         shift_labels
    )
```

table is an XRT/table widget; *position* is the first row number to delete; *num_rows* is the number of rows to be deleted; *shift_labels* specifies whether to shift up the labels of rows below the deleted rows. If the current cell is at or below *position*, the edit will be cancelled, and the XmText widget will be unmapped.

XrtTblDestroyCellValues()

Destroys an `XrtTblCellValues` structure and frees its memory.

```
void
XrtTblDestroyCellValues(
    XrtTblCellValues *cells
    )
```

cells may be `NULL`.

XrtTblDrawPS()

Generates encapsulated PostScript output of a table image, using PostScript fonts and device-independent PostScript operators. This method is identical to *XrtTblVaDrawPS()* but takes an arglist. *XrtTblDrawPS()* returns the number of pages output. See *XrtTblVaDrawPS()* on page 214 for a full description of the arguments.

```
int
XrtTblDrawPS(
    Widget          table,
    FILE            *file,
    XrtTblPSarg     *arglist,
    int             argcount
    )
```

table is an XRT/table widget; *file* should point to a file that is already open and ready for writing; *arglist* is a list of *argcount* arguments.

The following example shows the use of *XrtTblDrawPS()*:

```
int ac, num_pages;
double scale = FIT_TO_PAGE_WIDTH;
XrtTblPSarg al[20];

ac = 0;
XtSetArg(al[ac], XRTTBL_PS_SCALE, &scale); ac++;
num_pages = XrtTblDrawPS(table, fp, al, ac);
```

XrtTblEnableXrtField()

Specifies that an XRT/field widget is to be created to edit any cell which has an `XmNxrtTblDataTypeContext` value corresponding to an XRT/field data type (such as `XRTTBL_TYPE_FIELD_STRING`). This procedure must be called before any table is created; otherwise, a warning is issued and an XmText widget is used. The XRT/field library (`libxrtfield.a`) must appear in the link line before the XRT/table library (`libxrttable.a`).

```
void
XrtTblEnableXrtField(void)
```

XrtTblEventToRowCol()

Determines the cell or label in which an X event occurred. *XrtTblEventToRowCol()* returns `False` if the event was not within a cell or label. This method is usually called from an action routine.

```
Boolean
XrtTblEventToRowCol(
    Widget      table,
    XEvent      *event,
    int         *row,       /* Returned */
    int         *column     /* Returned */
    )
```

table is an XRT/table widget; *row* and *column* is the index of the cell or label.

XrtTblFlash()

Flashes a *context* of cells and/or labels. The cells flash for a specified duration, using the colors you specify. *XrtTblFlash()* returns `False` if any argument is invalid.

```
Boolean
XrtTblFlash(
    Widget          table,
    XrtTblContext   *context,
    int             count,
    String          bg_color,
    String          fg_color,
    unsigned long   on_duration,
    unsigned long   off_duration
    )
```

table is an XRT/table widget; *context* specifies the part of the table to flash; *count* is the number of times to flash (`XRTTBL_MAXINT` flashes an infinite number of times); *bg_color* and *fg_color* specify the flash background and foreground colors (`NULL` uses the cell's "opposite" color); *on_duration* and *off_duration* specify the minimum duration of each part of the flash in milliseconds (0 specifies a duration of 500 milliseconds for that half of the flash).

Cells with the same *on_duration* and *off_duration* values flash synchronously.

XrtTblGet3dData() (source)

Creates an `XrtTblCellValues` structure and loads it with data converted from an XRT/3d dataset. *XrtTblGet3dData()* returns `NULL` if any argument is invalid.

```
XrtTblCellValues *
XrtTblGet3dData(
    Xrt3dData       *data,
    String          format,
    )
```

data specifies a regular or irregular grid XRT/3d dataset, but not the scatter plot dataset; *format* is a *sprintf* format string (such as "`%.2f`"), which is used to convert numeric data from type double to type String.

XrtTblGetCell()

Gets the value from table, if present, or from `XmNxrtTblCelValueFunc` or `XmNxrtTblCellValueCallback`. A string is returned if `XmNxrtTblXmStringSetMode` is `XRTTBL_SET_STRING`.

```
XmString
XrtTblGetCell(
    Widget      table,
    int         row,
    int         column
    )
```

table is an XRT/table widget; *row* and *column* specify the cell or label index. Do not free the returned XmString.

XrtTblGetCellDimensions()

Determines the size of a cell or label.

```
Boolean
XrtTblGetCellDimensions(
    Widget      table,
    int         row,
    int         column,
    int         *pixel_width,   /* Returned */
    int         *pixel_height,  /* Returned */
    int         *char_width,    /* Returned */
    int         *char_height    /* Returned */
    )
```

table is an XRT/table widget; *row* and *column* specify the cell or label index; *pixel_height* and *pixel_width* are the height and width of the cell or label in pixels; *char_height* and *char_width* are the height and width of the cell or label calculated in characters.

XrtTblGetCellValues()

Creates a new `XrtTblCellValues` structure and loads it with values copied from the `XrtTblCellValues` structure pointed to by `XmNxrtTblCellValues`. *XrtTblGetCellValues()* returns `NULL` if any argument is invalid. The returned structure should be freed after use with *XrtTblDestroyCellValues()*.

```
XrtTblCellValues *
XrtTblGetCellValues(
    Widget      table,
    int         start_row,
    int         start_column,
    int         end_row,
    int         end_column,
    Boolean     convert_to_String,
    )
```

table is an XRT/table widget; *start_row* and *start_column* specify the location of the starting cell in the range; *end_row* and *end_column* specify the location of the ending cell in the range. The XmString values are converted to Strings if *convert_to_String* is `True`.

XrtTblGetCellValuesSize()

Determines the row number of the last non-blank row, and the longest row in an `XrtTblCellValues` structure. *XrtTblGetCellValuesSize()* returns `NULL` if the structure is empty.

```
Boolean
XrtTblGetCellValuesSize(
    XrtTblCellValues    *cells,
    int                 *last_row,          /* Returned */
    int                 *last_column,       /* Returned */
    int                 *longest_row        /* Returned */
    )
```

last_row is the last non-blank row; *last_column* is the length of the row (minus 1) at the position specified by *longest_row*.

XrtTblGetCurrentCell()

Determines the location of the current cell, or returns `False` if there is no current cell.

```
Boolean
XrtTblGetCurrentCell(
    Widget      table,
    int         *row,       /* Returned */
    int         *column     /* Returned */
    )
```

table is an XRT/table widget; *row* and *column* specify the cell index.

XrtTblGetGraphData() (source)

Creates an `XrtTblCellValues` structure and loads it with data converted from an XRT/graph dataset. This procedure handles Array datasets only. *XrtTblGetGraphData()* returns `NULL` if any argument is invalid.

```
XrtTblCellValues *
XrtTblGetGraphData(
    Void        *data
    String      format,
    )
```

data is an XRT/graph dataset (you can pass either a XRT/graph data object handle or a pointer to an `XrtData` structure); *format* is a *sprintf* format string (such as "`%.2f`"), which is used to convert data from type float to type String.

XrtTblGetPreviewProperties()

Retrieves the appearance attributes of the preview scrolling window.

```
void
XrtTblGetPreviewProperties(
    Widget              table,
    Pixel               foreground,
    Pixel               background,
    XmFontList          *fontlist
    XrtTblBorderType    *border);
```

All values are returned, but *fontlist* is not copied and should not be freed.

XrtTblGetRowColByWidget()

Determines the location and parent table of any widget set in a cell/label (specified with `XmNxrtTblWidgetContext`). This widget may be a child of a widget in a cell. *XrtTblGetRowColByWidget()* returns `False` if the widget is not within a cell or label, or if any argument is invalid.

```
Boolean
XrtTblGetRowColByWidget(
    Widget      widget,
    Widget      *table,     /* Returned */
    int         *row,       /* Returned */
    int         *column     /* Returned */
    )
```

table is the parent XRT/table widget; *row*, *column* specify the location of the widget in the table.

XrtTblGetRowColLocation()

Determines the pixel coordinates of the top-left corner of a cell or label. *XrtTblGetRowColLocation()* returns `False` if any argument is invalid.

```
Boolean
XrtTblGetRowColLocation(
    Widget      table,
    int         row, column,
    Window      *window,        /* Returned */
    int         *table_x,       /* Returned */
    int         *table_y,       /* Returned */
    int         *window_x,      /* Returned */
    int         *window_y       /* Returned */
    )
```

table is an XRT/table widget; *window* is the parent window of the cell/label; *table_x* and *table_y* are the co-ordinates with respect to the table; *window_x* and *window_y* are the co-ordinates with respect to the parent window.

XrtTblGetSelectedRange()

Retrieves a currently-selected range from the `XrtTblRange` structure pointed to by `XmNxrtTblSelectedCellList`. *XrtTblGetSelectedRange()* returns `False` if no cells are selected, or any argument is invalid.

```
Boolean
XrtTblGetSelectedRange(
    Widget          table,
    int             position,
    XrtTblRange     *range      /* Returned */
    )
```

table is an XRT/table widget; *position* is the position within the selected cell list; *range* is returned containing the selected range.

The range returned by *XrtTblGetSelectedRange()* is rationalized to read from top to bottom and from left to right. Special values such as `XRTTBL_MAXINT` are converted to valid range values.

XrtTblGetSpanByRowCol()

Determines the spanned range that includes a specified cell or label. *XrtTblGetSpanByRowCol()* returns `False` if the cell/label is not part of any spanned range, or if any argument is invalid.

```
Boolean
XrtTblGetSpanByRowCol(
    Widget          table,
    int             row, column,
    XrtTblRange     *range      /* Returned */
    )
```

table is an XRT/table widget; *row* and *column* specify the cell/label location; *range* is returned containing the spanned range.

XrtTblGetWidgetByRowCol()

Returns the widget currently displayed in a cell or label. *XrtTblGetWidgetByRowCol()* returns `NULL` if there is no widget in the cell or label or if the widget is not currently displayed.

```
Widget
XrtTblGetWidgetByRowCol(
    Widget      table,
    int         row, column
    )
```

table is an XRT/table widget; *row* and *column* specify the cell/label location.

XrtTblGetXrtFieldByType()

Finds a previously-created XRT/field widget child of the specified type, or creates one if necessary.

```
Widget
XrtTblGetXrtFieldByType(
    Widget          table,
    XrtTblDataType  type
    )
```

table is an XRT/table widget; *type* is an XRT/field widget type. The elements of `XrtTblDataType` are the same as XRT/field's `XrtFldFieldType` data type.

XrtTblImportExportListAdd()

Adds an item to a list of drag and drop target items.

```
XrtTblImportExportStruct **
XrtTblImportExportListAdd(
    XrtTblImportExportStruct    **ie_list,
    int                         position,
    XrtTblImportExportStruct    *importExportElement
    )
```

ie_list is the target item list to be added to. *importExportElement* is the new target item to be added. *position* indicates the location of the new item in the list; set this to 0 to just add to the front of the list. The updated target item list is returned.

XrtTblImportExportListCopy()

Copies a list of drag and drop target items.

```
XrtTblImportExportStruct **
XrtTblImportExportListCopy(
    XrtTblImportExportStruct **ie_list,
    )
```

ie_list is the target item list to be copied. The copy is returned.

XrtTblImportExportListDelete()

Deletes an item from a drag and drop target item list.

```
XrtTblImportExportStruct **
XrtTblImportExportListDelete(
    XrtTblImportExportStruct    **ie_list,
    String                      target
    )
```

ie_list is the target item list to be deleted from. *target* is the target item to be deleted. The updated target item list is returned.

XrtTblImportExportListDestroy()

Destroys a drag and drop target item list.

```
void
XrtTblImportExportListDestroy(
    XrtTblImportExportStruct **ie_list,
    )
```

ie_list is the target item list to be destroyed.

XrtTblIsCellSelected()

Determines whether the specified cell is selected. If it is, this method returns `True` and gets the range and position in the selected cell list.

```
Boolean
XrtTblIsCellSelected(
    Widget          table,
    int             row, column,
    int             *range_index, /* Returned */
    XrtTblRange     *range        /* Returned */
    )
```

table is an XRT/table widget; *row* and *column* specify the cell location; *range_index* is the position within the selected cell list; *range* specifies the selected range itself.

XrtTblIsCellTraversable()

Returns `True` if the specified cell is currently traversable. A cell is traversable if `XmNxrtTblTraversableContext` is `True`; if a widget is in the cell, the cell is traversable if `XmNtraversalOn` is `True`. Labels are never traversable.

```
Boolean
XrtTblIsCellTraversable(
    Widget      table,
    int         row, column
    )
```

table is an XRT/table widget; *row* and *column* specify the cell location.

XrtTblIsCellVisible()

Returns `True` if the specified cell or label is currently visible on the screen.

```
Boolean
XrtTblIsCellVisible(
    Widget      table,
    int         row, column
    )
```

XrtTblMakeCellVisible()

Makes the specified cell or label visible, scrolling the table if necessary. *XrtTblMakeCellVisible()* returns `False` if any argument is invalid.

```
Boolean
XrtTblMakeCellVisible(
    Widget      table,
    int         row, column
    )
```

XrtTblMoveColumns()

Moves a range of columns. *XrtTblMoveColumns()* returns `False` if any argument is invalid. All attributes (such as the background color) move along with the columns; the values of all Series resources are adjusted as necessary.

```
Boolean
XrtTblMoveColumns(
    Widget      table,
    int         source,
    int         num_columns,
    int         destination,
    Boolean     shift_labels
    )
```

table is an XRT/table widget; *source* is the column number of the left side of the range; *num_columns* is the number of columns to move; *destination* is the column number before which to insert the moved columns; *shift_labels* specifies whether to move the column labels too. If the current cell is at or to the right of *Min*(*source*, *destination*), the edit will be cancelled, and the XmText widget will be unmapped.

XrtTblMoveRows()

Moves a range of rows. *XrtTblMoveRows()* returns `False` if any argument is invalid. All attributes (such as the background color) move along with the rows; the values of all Series resources are adjusted as necessary.

```
Boolean
XrtTblMoveRows(
    Widget      table,
    int         source,
    int         num_rows,
    int         destination,
    Boolean     shift_labels
    )
```

table is an XRT/table widget; *source* is the row number of the top row of the range; *num_rows* is the number of rows to move; *destination* is the row number before which to insert the moved rows; *shift_labels* specifies whether to move the row labels too. If the current cell is at or below *Min*(*source*, *destination*), the edit will be cancelled, and the XmText widget will be unmapped.

XrtTblMrmInitialize()

Registers the XRT/table widget class with Mrm, for use by UIL applications. This procedure must be called after *MrmInitialize()*.

```
int
XrtTblMrmInitialize(void)
```

XrtTblRangeListAdd()

Inserts or adds a range to a range list, returning the enlarged list. The last element will be set to `NULL`. The returned range list should be freed after use with *XrtTblRangeListFree()*.

```
XrtTblRange **
XrtTblRangeListAdd(
    XrtTblRange     **list,
    int             position,
    XrtTblRange     *range
    )
```

list specifies the source range list (if `NULL`, a new list is created, ignoring position); *position* is the range index at which to insert *range* (if `XRTTBL_MAXINT`, *range* is added after the last range in the list); *range* is the range to add to the list.

XrtTblRangeListCopy()

Copies a range list (such as the selected range list specified by `XmNxrtTblSelectedCellList`). The returned range list should be freed after use with *XrtTblRangeListFree()*.

```
XrtTblRange **
XrtTblRangeListCopy(
    XrtTblRange     **list
    )
```

list specifies a list of ranges.

XrtTblRangeListDelete()

Deletes a range from a range list, returning the smaller list. The returned range list should be freed after use with *XrtTblRangeListFree()*.

```
XrtTblRange **
XrtTblRangeListDelete(
    XrtTblRange     **list,
    int             position
    )
```

list specifies a list of ranges*; position* is the range index of the range to delete.

XrtTblRangeListFind()

Finds the provided row and column in the provided list of ranges, returning the index of the first range found containing the provided row and column. If no match is found, `XRTTBL_NOTFOUND` is returned.

```
int
XrtTblRangeListFind(
    XrtTblRange     **range,
    int             row,
    int             column
    )
```

range specifies the range(s) to search; *row* and *column* is the cell to search for.

XrtTblRangeListFindRange()

Finds the provided range in the provided list of ranges and returns the index of the first match. If no match is found, `XRTTBL_NOTFOUND` is returned.

```
int
XrtTblRangeListFindRange(
    XrtTblRange     **range
    XrtTblRange     *request
    )
```

range specifies the range(s) to search; *request* specifies the range to search for.

XrtTblRangeListFree()

Frees a range list.

```
void
XrtTblRangeListFree(
    XrtTblRange     **list
    )
```

list specifies a list of ranges.

XrtTblReadAscii() (source)

Creates a new `XrtTblCellValues` structure and loads it with data read from an ASCII file. The file should contain one row of delimited data per line. *XrtTblReadAscii()* returns `NULL` if an error occurs.

```
XrtTblCellValues *
XrtTblReadAscii(
    Widget      table,
    FILE        *file,
    char        delimiter
    )
```

table is an XRT/table widget; *file* points to a file that is opened for read access; *delimiter* is the character that appears between the cell values in each row. A backslash in the file will escape the delimiter character.

XrtTblReadResources()

Reads a resource file and sets table resources to the values in the file. This method returns `False` if an error occurs.

```
Boolean
XrtTblReadResources(
    Widget      table,
    String      filename
    )
```

table is an XRT/table widget; *filename* is the resource file.

XrtTblRemoveAllCallbacks()

Removes all the callbacks from an XRT/table callback list. This method is equivalent to *XtRemoveAllCallbacks()*, except that it notifies the table that the callbacks have been removed.

```
void
XrtTblRemoveAllCallbacks(
    Widget      table,
    String      cb_name
    )
```

table is an XRT/table widget; *cb_name* is the resource name of the callback list.

XrtTblRemoveCallback()

Removes a callback from an XRT/table callback list. This method is equivalent to *XtRemoveCallback()*, except that it notifies the table that a callback has been removed.

```
void
XrtTblRemoveCallback(
    Widget              table,
    String              cb_name,
    XtCallbackProc      cb
    )
```

table is an XRT/table widget; *cb_name* is the resource name of the callback list; *cb* is the callback to be removed.

XrtTblRemoveCallback() must be used when removing callbacks from `XmNxrtTblCellValueCallback` and `XmNxrtTblLabelCreateCallback` lists.

XrtTblRepaintRange()

Redraws a range of cells and/or labels if `XmNxrtTblRepaint` is `XRTTBL_REPAINT_ON`. No Expose callbacks are triggered. *XrtTblRepaintRange()* returns `False` if any argument is invalid.

```
Boolean
XrtTblRepaintRange(
    Widget          table,
    XrtTblRange     *range
    )
```

table is an XRT/table widget. *range* specifies the range to repaint; if `NULL`, the entire table is redrawn.

This method only needs to be called when `XmNxrtTblCellValueCallback` or `XmNxrtLabelCreateCallback` has changed and the application needs to force a redraw.

If the current cell is within the specified range, the XmText widget containing the edited value is reset (by copying from the `XrtTblCellValues` structure or calling the `XmNxrtTblCellValueCallback` method).

XrtTblRestoreAllCursors()

Restores all the cursors in XRT/table to their default values.

```
void
XrtTblRestoreAllCursors(
    Widget          table
    )
```

table is an XRT/table widget.

XrtTblSeriesAdd()

Adds a member to a series for a context (previously created with *XrtTblSeriesCreate()*). If the context specified already exists in the series, it is replaced.

```
void
XrtTblSeriesAdd(
    XrtTblSeries    *series,
    XrtTblContext   *context,
    XtArgVal        value
    )
```

series is the series to be added to; *context* specifies the context of the new series member; *value* specifies the new value.

XrtTblSeriesAddValue()

Adds a member to a series at a series position (previously created with *XrtTblSeriesCreate()*). If the position already exists in the series, it is replaced.

```
void
XrtTblSeriesAddValue(
    XrtTblSeries    *series,
    int             position,
    XtArgVal        value
    )
```

series is the series to be added to; *position* specifies the index in the series (added to the end if `XRTTBL_MAXINT`); *value* specifies the new value.

XrtTblSeriesCopy()

Copies a series.

```
XrtTblSeries *
XrtTblSeriesCopy(
    XrtTblSeries    *series
    )
```

series is the series to be copied.

XrtTblSeriesCreate()

Creates a series.

```
XrtTblSeries *
XrtTblSeriesCreate(
    XrtTblValueType     type
    )
```

type specifies the series type to create.

XrtTblSeriesDelete()

Deletes a member from a series.

```
void
XrtTblSeriesDelete(
    XrtTblSeries    *series,
    int             position
    )
```

series is the series to be deleted from; *position* specifies the index of the member to delete.

XrtTblSeriesDestroy()

Destroys a series and frees its memory.

```
void
XrtTblSeriesDestroy(
    XrtTblSeries        *series
    )
```

series is the series to be destroyed.

XrtTblSeriesFind()

Finds a cell or label within a series. *XrtTblSeriesFind()* returns the index of the member within the series that matches, or `XRTTBL_NOTFOUND` if no match is found.

```
int
XrtTblSeriesFind(
    XrtTblSeries        *series,
    int                 row,
    int                 column
    )
```

series is the series to be searched; *row* and *column* specify the cell or label to be found.

XrtTblSeriesIterate()

Used to examine the members in a series, this procedure returns a series member at the specified position, and increments the position counter. *XrtTblSeriesIterate()* returns `False` if the position is invalid.

```
Boolean
XrtTblSeriesIterate(
    XrtTblSeries    *series,
    int             *position,
    int             *row,          /* Returned */
    int             *column,       /* Returned */
    XtArgVal        *value         /* Returned */
    )
```

position is the member within the series (should be initialized to 0); *row* and *column* is the cell index of the cell found; *value* is the value at the position.

XrtTblSeriesSetValue()

Sets a series member at a given position within a series. This procedure allocates and adds a new series member if the position is greater than the position of the last series member.

```
void
XrtTblSeriesSetValue(
    XrtTblSeries        *series,
    int                 position,
    XrtTblContext       *context,
    XtArgVal            value
    )
```

position is the member within the series; *context* is the table context; *value* is the value to be set for the series member.

XrtTblSetAllCursors()

Sets all the cursors in XRT/table to the provided cursor.

```
void
XrtTblSetAllCursors(
    Widget          table
    Cursor          cursor
    )
```

table is an XRT/table widget; *cursor* is the provided cursor to apply to the widget.

XrtTblSetCell()

Sets the value of the indicated cell, optionally validating and redrawing the cell. Though the data type of *value* is XmString, you can specify a String if linking to Motif 1.2. Strings can also be specified if `XmNxrtTblXmStringSetMode` is set appropriately.

```
void
XrtTblSetCell(
    Widget      table,
    int         row,
    int         column,
    XmString    value,
    Boolean     validate,
    Boolean     redraw
    )
```

table is an XRT/table widget; *row* and *column* is the cell to set; *value* is the value to be set into the cell; *validate* is whether or not to validate *value*; *redraw* is whether or not to redraw the cell after it has been set.

This routine does not resize the cell to fit the new data, even if `XmNxrtTblPixelWidthContext` is `XRTTBL_VARIABLE`.

XrtTblSetCellValues()

Copies values from a supplied `XrtTblCellValues` structure to the `XrtTblCellValues` structure pointed to by `XmNxrtTblCellValues`. *XrtTblSetCellValues()* returns `False` if any of the arguments is invalid.

```
Boolean
XrtTblSetCellValues(
    Widget              table,
    int                 start_row,
    int                 start_column,
    XrtTblCellValues    *values,
    Boolean             resize
    )
```

table is an XRT/table widget; *start_row* and *start_column* specify the cell in the receiving `XrtTblCellValues` structure at which to begin the copy; *values* is the source `XrtTblCellValues` structure; *resize* specifies whether to increase (if necessary) the number of rows and columns in the table.

XrtTblSetContext()

Defines a context for use when specifying the `XmNxrtTblContext` resource. Since this procedure returns a pointer to static storage, the pointer must not be freed.

```
XrtTblContext *
XrtTblSetContext(
    int         row,
    int         column
    )
```

row and *column* specify the context. In addition to a cell index, *row* and/or *column* can also be assigned special values:

`XRTTBL_ALL`	all cells and labels in a row or column
`XRTTBL_LABEL`	a row or column label
`XRTTBL_ALLCELLS`	all cells (but not labels) in a row or column

XrtTblSetPreviewProperties()

Determines the appearance of the preview scrolling window. Initial values are derived from the table's attributes.

```
void
XrtTblSetPreviewProperties(
    Widget              w,
    Pixel               foreground,
    Pixel               background,
    XmFontList          fontlist,/*This is copied internally*/
    XrtTblBorderType    border);
```

border is `XRTTBL_BORDER_NONE` by default. *foreground* and *background* are `XmNforeground` and `XmNbackground` from XRT/table. *fontlist* is XRT/table's default fontlist.

XrtTblSetPSarg()

Sets a resource name and value in an `XrtTblPSArg` list.

```
void
XrtTblSetPSarg(
    XrtTblPSarg         arg,
    XrtTblDrawPSarg     name,
    XtArgVal            value
    )
```

arg is the arglist; *name* specifies an argument name; *value* is the value set for *name*. If the data type of *name* is a float (for example, `XRTTBL_PS_SCALE`), its value must be passed as a pointer.

XrtTblSetRange()

Creates a range context for an `XmNxrtTblRangeContext` resource value. Since this procedure returns a pointer to static storage, the pointer must not be freed.

```
XrtTblRange *
XrtTblSetRange(
    int     start_row, start_column,
    int     end_row, end_column
    )
```

XrtTblSetReverseConverters() (source)

Registers XRT/table's reverse resource converters for use with *XtConvertAndStore()*. Reverse converters allow applications to get resource values in a human-readable form.

```
void
XrtTblSetReverseConverters(void)
```

XrtTblSetXmStringConverter()

Sets the `XmRXmString` resource converter to create XmString values for cells and labels. This procedure must be called before the table is created.

```
void
XrtTblSetXmStringConverter(void)
```

The conversion uses the following syntax to specify embedded font changes (tags):

@font@	font's tag in `XmNfontList`
@P@	revert to previous font
\@	a literal "@"

For example, the string "`For a @bold@free evaluation@P@ of any product`" displays[1] as

For a **free evaluation** of any product

1. The fontlist is "fixed,-*-times-bold-r-normal-*-140-*=bold".

XrtTblSortTable()

Sorts the rows in the table's `XrtTblCellValues` structure based on a specified column. This method uses an application-supplied sorting procedure to sort the table. *XrtTblSortTable()* does not sort cell values supplied by a cell value method (`XmNxrtTblCellValueCallback`). The widget must be realized.

```
Boolean
XrtTblSortTable(
    Widget      table,
    int         column,
    int         (* compare)
                (void *, void *)
    )
```

table is an XRT/table widget; *column* is the index on which to base the sort. *compare* is a procedure that is passed values from two cells in the column, and returns -1 (less than), 0 (equal to), or 1 (greater than) after comparing the first value with the second (see *qsort(3)*). If *compare* is `NULL`, the data will by default be sorted in ascending alphabetical order using the following procedure:

```
int
sort_compare(
    XrtTblCompareStruct     *row1,
    XrtTblCompareStruct     *row2
    )
{
    if (row1->string && row2->string)
        return (strcmp(row1->string, row2->string));
    else if (row1->string)
        return (-1);
    else if (row2->string)
        return (1);
    else
        return (0);
}
```

`XrtTblCompareStruct` is defined as:

```
typedef struct {
    Widget      table;
    int         row, column;
    String      string;
} XrtTblCompareStruct *;
```

Calling *XrtTblSortTable()* has no effect on the spanned ranges, cell resources or selected cells of the table, as only the values stored in the table are changed.

To be notified of the new row order, use the `XmNxrtTblSortCallback` resource.

XrtTblSyncScrolling()

Synchronizes the scrolling of multiple tables or multiple scrollbars horizontally or vertically. If the scrollbars are of different length, the tables will be scrolled

proportionally. Scrolling is symmetric; scrolling one table or scrollbar also scrolls the other. *XrtTblSyncScrolling()* returns `False` if any argument is invalid.

```
Boolean
XrtTblSyncScrolling(
    Widget      my_widget1,
    Widget      my_widget2,
    int         orientation
    )
```

my_widget1 and *my_widget2* are either two table widgets or two XmScrollBar widgets; *orientation* is `XmHORIZONTAL`, `XmVERTICAL` or `XmNONE`.

XrtTblTraverseToCell()

Attempts to move the current cell to a new location. *XrtTblCommitEdit()* is first called to commit the edit in the current cell. If *XrtTblCommitEdit()* returns `False`, *XrtTblTraverseToCell()* returns `False` and does not move the XmText widget. Otherwise, *XrtTblTraverseToCell()* attempts to traverse to the specified cell. If the cell is not traversable, the method returns `False`. The callbacks in the `XmNxrtTblEnterCellCallback` list are called for the specified cell to allow it to set the XmText widget's editability, and the XmText widget is then mapped on top of the cell.

```
Boolean
XrtTblTraverseToCell(
    Widget      table,
    int         row,
    int         column,
    Boolean     select
    )
```

table is an XRT/table widget; *row* and *column* specify the index of the cell to be traversed to; *select* specifies whether the cell should be selected (that is, whether the `XmNxrtTblSelectCallback` routines should be invoked).

XrtTblVaDrawPS()

Generates encapsulated PostScript (EPSF-2.0) output of a table image, using PostScript fonts and device-independent PostScript operators. *XrtTblVaDrawPS()* uses a *varargs*-style interface similar to *XtVaSetValues()*, using attribute-value pairs of type `XrtTblVaDrawPS`. *XrtTblVaDrawPS()* returns the number of pages output (0 pages indicates an error). See section 3.18 on page 65 for complete information on printing tables.

```
int
XrtTblVaDrawPS(
    Widget      table,
    FILE        *file,
    ...,
    )
```

table is an XRT/table widget; *file* should point to a file that is already open and ready for writing. The following summarizes the attributes and values:

Attribute	Values/Type	Default
XRTTBL_PS_CELL_RANGE	XrtTblRange *	NULL (the entire table)
XRTTBL_PS_COLOR	XRTTBL_PS_AS_IS XRTTBL_PS_MONO XRTTBL_PS_AUTO	XRTTBL_PS_AS_IS
XRTTBL_PS_COL_PAGEBREAKS	int *	NONE (breaks as needed)
XRTTBL_PS_FOOTER	string	NULL
XRTTBL_PS_FOOTER_FONT	string	"Courier 12"
XRTTBL_PS_FOOTER_MARGIN	double	see description
XRTTBL_PS_FOOTER_POINT_SIZE	double	12.0
XRTTBL_PS_GRID_TYPE	XRTTBL_PS_AS_IS XRTTBL_PS_LINE XRTTBL_PS_NONE	XRTTBL_PS_AS_IS
XRTTBL_PS_HEADER	string	NULL
XRTTBL_PS_HEADER_FONT	string	"Courier 12"
XRTTBL_PS_HEADER_MARGIN	double	see description
XRTTBL_PS_HEADER_POINT_SIZE	double	12.0
XRTTBL_PS_MARGIN_BOTTOM	double	see description
XRTTBL_PS_MARGIN_LEFT	double	see description
XRTTBL_PS_MARGIN_RIGHT	double	see description
XRTTBL_PS_MARGIN_TOP	double	see description
XRTTBL_PS_NUM_COPIES	int	1
XRTTBL_PS_ORIENTATION	XRTTBL_PS_PORTRAIT XRTTBL_PS_LANDSCAPE XRTTBL_PS_BEST	XRTTBL_PS_PORTRAIT
XRTTBL_PS_PAPERSIZE_WIDTH	double	8.5
XRTTBL_PS_PAPERSIZE_HEIGHT	double	11.0
XRTTBL_PS_ROW_PAGEBREAKS	int *	NONE (breaks as needed)
XRTTBL_PS_SCALE	double FIT_TO_PAGE FIT_TO_PAGE_WIDTH FIT_TO_PAGE_HEIGHT	72.0
XRTTBL_PS_SHOWPAGE	Boolean	True
XRTTBL_PS_SHOW_COL_LABELS	XRTTBL_PS_EDGE_PAGES XRTTBL_PS_ALL XRTTBL_PS_NONE	XRTTBL_PS_EDGE_PAGES

Attribute	Values/Type	Default
XRTTBL_PS_SHOW_FROZEN_COLS	XRTTBL_PS_EDGE_PAGES XRTTBL_PS_ALL XRTTBL_PS_NONE	XRTTBL_PS_EDGE_PAGES
XRTTBL_PS_SHOW_FROZEN_ROWS	XRTTBL_PS_EDGE_PAGES XRTTBL_PS_ALL XRTTBL_PS_NONE	XRTTBL_PS_EDGE_PAGES
XRTTBL_PS_SHOW_ROW_LABELS	XRTTBL_PS_EDGE_PAGES XRTTBL_PS_ALL XRTTBL_PS_NONE	XRTTBL_PS_EDGE_PAGES
XRTTBL_PS_SHOW_PIXMAPS	XRTTBL_PS_ALL XRTTBL_PS_NONE	XRTTBL_PS_ALL
XRTTBL_PS_SHOW_TITLES	XRTTBL_PS_EDGE_PAGES XRTTBL_PS_ALL XRTTBL_PS_NONE	XRTTBL_PS_EDGE_PAGES
XRTTBL_PS_SHOW_WIDGETS	XRTTBL_PS_ALL XRTTBL_PS_NONE	XRTTBL_PS_ALL
XRTTBL_PS_UNITS	XRTTBL_PS_INCHES XRTTBL_PS_CM	XRTTBL_PS_INCHES

XRTTBL_PS_CELL_RANGE specifies the starting and ending row and column to be output. By default, the entire table is printed.

XRTTBL_PS_COLOR specifies how colors are handled. When defaulted to XRTTBL_PS_AS_IS, the table is output in color if displayed on a color X Server or in monochrome if displayed on a monochrome X Server. Color PostScript files can be printed on a black and white printer; the printer translates each color to a shade of gray. When set to XRTTBL_PS_MONO, all foreground colors are set to black, all background colors are set to white, and all cell shadows (if printed) are set to gray. When set to XRTTBL_PS_AUTO, the PostScript file determines at print time whether it is printing on a color or monochrome printer, and prints the image accordingly (on most printers).

XRTTBL_PS_COL_PAGEBREAKS and *XRTTBL_PS_ROW_PAGEBREAKS* are lists of integers (terminated by -1) that specify where to break each page. Each integer is the index of a row or column that should begin a new page. If no page breaks are specified, page breaks are determined automatically.

XRTTBL_PS_FOOTER, *XRTTBL_PS_FOOTER_FONT*, *XRTTBL_PS_HEADER*, and *XRTTBL_PS_HEADER_FONT* allow special characters to be embedded, as in a commercial spreadsheet application. By default, headers and footers are left-aligned, but can be center or right-aligned by using the "|" character. The first occurrence of "|" centers the text which follows. A second occurrence right-justifies the text which follows. Other symbols:

#	current page number
^	total number of pages

%<*x*> current date in *strftime* format, where <*x*> is any *strftime* formatting symbol

\ next character is a literal

XRTTBL_PS_FOOTER_MARGIN is only defined if *XRTTBL_PS_FOOTER* is set. It specifies the margin between the footer and the edge of the page. Its default, if a footer is specified, is 0.25 units (set by *XRTTBL_PS_UNITS*).

XRTTBL_PS_GRID_TYPE specifies how to draw the cell/label borders. When defaulted to `XRTTBL_PS_AS_IS`, borders are drawn as they appear on the screen. When `XRTTBL_PS_LINE`, borders are drawn as simple thin black ruling lines. When `XRTTBL_PS_NONE`, no borders are drawn.

XRTTBL_PS_HEADER_MARGIN is only defined if *XRTTBL_PS_HEADER* is set. It specifies the margin between the header and the edge of the page. Its default, if a header is specified, is 0.25 units (set by *XRTTBL_PS_UNITS*).

XRTTBL_PS_MARGIN_TOP and *XRTTBL_PS_MARGIN_BOTTOM* specify the width of the non-printing boundary around the "top" and "bottom" edges of the *paper* (*not* around the top and bottom of the image). The default value for these attributes is 0.25 units (set by *XRTTBL_PS_UNITS*); however:

- If *XRTTBL_PS_HEADER* is set and *XRTTBL_PS_ORIENTATION* is set to `XRTTBL_PS_PORTRAIT` (or the orientation is set to portrait mode by `XRTTBL_PS_BEST`), the default value for *XRTTBL_PS_MARGIN_TOP* is 0.75 units.
- If *XRTTBL_PS_FOOTER* is set and *XRTTBL_PS_ORIENTATION* is set to `XRTTBL_PS_PORTRAIT` (or the orientation is set to portrait mode by `XRTTBL_PS_BEST`), the default value for *XRTTBL_PS_MARGIN_BOTTOM* is 0.75 units.

XRTTBL_PS_MARGIN_LEFT and *XRTTBL_PS_MARGIN_RIGHT* specify the width of the non-printing boundary around the "left" and "right" edges of the *paper* (*not* around the top and bottom of the image). The default value for these attributes is 0.25 units (set by *XRTTBL_PS_UNITS*); however:

- If *XRTTBL_PS_HEADER* is set and *XRTTBL_PS_ORIENTATION* is set to `XRTTBL_PS_LANDSCAPE` (or the orientation is set to landscape mode by `XRTTBL_PS_BEST`), the default value for *XRTTBL_PS_MARGIN_LEFT* is 0.75 units.
- If *XRTTBL_PS_FOOTER* is set and *XRTTBL_PS_ORIENTATION* is set to `XRTTBL_PS_LANDSCAPE` (or the orientation is set to landscape mode by `XRTTBL_PS_BEST`), the default value for *XRTTBL_PS_MARGIN_RIGHT* is 0.75 units.

XRTTBL_PS_NUM_COPIES specifies the number of copies to print. If the output is intended for inclusion in another document, this should be left at 1.

XRTTBL_PS_ORIENTATION specifies the orientation of the image on the paper. `XRTTBL_PS_LANDSCAPE` rotates the table image 90 degrees counter-clockwise. `XRTTBL_PS_BEST` causes *XrtTblVaDrawPS()* to compare the table range to the paper width and height, and select the best image orientation to use. For example, on a

standard 8.5 x 11 page, a range whose total width is greater than its total height will be output in landscape if XRTTBL_PS_ORIENTATION is set to XRTTBL_PS_BEST.

XRTTBL_PS_PAPERSIZE_WIDTH and *XRTTBL_PS_PAPERSIZE_HEIGHT* specify the width and height of the *paper* being used. The width and height do *not* change with the image orientation. Since paper always feeds through a printer oriented the same way, regardless of how the image on the page is oriented, the paper width and height attributes are tied to the "top" and "left" edges of the paper and not to the image. The number specified here is in the units set by XRTTBL_PS_UNITS. Common paper sizes:

- 8.5 x 11 inches (Letter)
- 8.5 x 14 inches (Legal)
- 21.0 x 29.7 centimeters (A4)
- 14.8 x 21.0 centimeters (A5)

XRTTBL_PS_SCALE specifies the scaling of the table image in pixels per inch. To approximate the screen image, set this attribute to the resolution of the screen (the default value of 72.0 is the most common screen resolution). To increase the image size to 200%, set this attribute to 36.0; to decrease the image to 50%, set it to 144.0. The following enumerated values can be used to specify other scaling behavior:

- FIT_TO_PAGE scales the entire range to fit on one page.
- FIT_TO_PAGE_WIDTH scales the range so it fits the width of a page.
- FIT_TO_PAGE_HEIGHT scales the range so it fits the height of a page.

XRTTBL_PS_SHOWPAGE specifies whether to put a PostScript *showpage* operator at the end of each page of the PostScript output. To print several table images on one page, make several calls to *XrtTblVaDrawPS()* using the same file pointer with XRTTBL_PS_SHOWPAGE set to False in all but the last call.

XRTTBL_PS_SHOW_COL_LABELS and *XRTTBL_PS_SHOW_ROW_LABELS* specify how to include labels in the PostScript output. XRTTBL_PS_ALL puts row/column labels on every page. XRTTBL_PS_NONE removes row/column labels from all pages, only printing those specified in the range. XRTTBL_PS_EDGE_PAGES includes labels only on the edge pages of a multipage output, treating the output as a huge table broken into multiple pages (if there is only one page, the effect is the same as XRTTBL_PS_ALL).

XRTTBL_PS_SHOW_FROZEN_COLS and *XRTTBL_PS_SHOW_FROZEN_ROWS* specify how to include frozen cells in the PostScript output. XRTTBL_PS_ALL puts frozen rows/columns on every page. XRTTBL_PS_NONE removes frozen rows/columns from all pages, only printing those specified in the range. XRTTBL_PS_EDGE_PAGES includes frozen rows/columns only on the edge pages of a multipage output, treating the output as a huge table broken into multiple pages (if there is only one page, the effect is the same as XRTTBL_PS_ALL).

XRTTBL_PS_SHOW_PIXMAPS specifies how to include pixmaps in the PostScript output. XRTTBL_PS_ALL includes any pixmap set for cells/labels in the print range. XRTTBL_PS_NONE removes all pixmaps from the output.

XRTTBL_PS_SHOW_TITLES specifies how to include table titles in the PostScript output. XRTTBL_PS_ALL puts all titles on every page. XRTTBL_PS_NONE removes all titles from all pages. XRTTBL_PS_EDGE_PAGES includes titles only on the edge pages of a multipage output, treating the output as a huge table broken into multiple pages (if there is only one page, the effect is the same as XRTTBL_PS_ALL).

XRTTBL_PS_SHOW_WIDGETS specifies how to include cell/label widgets in the PostScript output. XRTTBL_PS_ALL includes any widget set for cells/labels in the print range. XRTTBL_PS_NONE removes all widgets from the output.

XRTTBL_PS_UNITS specifies the measurement units for the PAPERSIZE_ and MARGIN_ attributes.

XrtTblWriteAscii() (source)

Writes an ASCII file that contains the values in a range of cells. The method converts the XmString value for each cell to a String. *XrtTblWriteAscii()* returns False if an error occurred.

```
Boolean
XrtTblWriteAscii(
    Widget          table,
    FILE            *file,
    char            delimiter,
    XrtTblRange     *range
    )
```

table is an XRT/table widget; *file* points to a file that should already be opened for write access; *delimiter* is the character to insert between the columns in each row; *range* specifies the cells to be output (NULL outputs all cell values).

XrtTblWriteResources()

Writes the current table resource values to a file, returning False if an error occurs.

```
Boolean
XrtTblWriteResources(
    Widget      table,
    FILE        *fp,
    String      prefix,
    Boolean     write_inherited
    )
```

prefix is the string to be written before each resource name (to allow the resources from multiple tables to be written to the same file); typically, this is set to the table name. If *prefix* is NULL, the prefix "*." is written. *write_inherited* specifies whether inherited resources such as XmNwidth should be written.

XrtTblWriteResources() does not dump context values when XrtTblContextValueCallback is set, due to the potentially large number of entries.

XrtTblWriteSylk()

Writes a SYLK-formatted file that contains the values in a range of cells. The method converts the XmString value for each cell to a String. *XrtTblWriteSylk()* returns `False` if an error occurred.

```
Boolean
XrtTblWriteSylk(
    Widget          table,
    FILE            *file,
    XrtTblRange     *range
    )
```

table is an XRT/table widget; *file* points to a file that should already be opened for write access; *range* specifies the cells to be output (`NULL` outputs all cell values).

XrtTblXYtoRowCol()

Determines the cell or label at a set of pixel coordinates. To translate table widget coordinates to root window coordinates, use *XtTranslateCoords(). XrtTblXYtoRowCol()* returns `False` if the coordinates are not within a cell or label.

```
Boolean
XrtTblXYtoRowCol(
    Widget      table,
    int         root_x, root_y,
    int         *row, *column       /* Returned */
    )
```

root_x and *root_y* specify the pixel coordinates, relative to the root window; *row*, *column* are the returned location of the cell or label.

XtIsXrtTable()

Returns `True` if a widget is an XRT/table widget or a subclass.

```
Boolean
XtIsXrtTable(
    Widget      table
    )
```

C

Macros

The following macros are provided for convenience in accessing portions of the table. They are defined in *$XRTHOME/include/Xm/XrtTable.h*.

Macro Name	Definition

`CELL_VALUE(cells, row, column)`	`CELL_VALUE_EXISTS(cells, row, column) \`
	`? cells->values[row]->list[column] \`
	`: NULL`

Returns an XmString containing a cell value, or `NULL` if there is no value for the cell. *cells* is an `XrtTblCellValues` structure; *row* and *column* are integers specifying the location of a single cell.	

`CELL_VALUE_EXISTS(cells, row, column)`	`cells && row >= 0 \`
	`&& row < cells->rows \`
	`&& cells->values[row] \`
	`&& column >= 0 \`
	`&& column < cells->values[row]->length`

Returns `True` if the cell contains a value, `False` if it does not. *cells* is an `XrtTblCellValues` structure; *row* and *column* are integers specifying the location of a single cell.	
`IS_CELL(row, column)`	`row >= 0 && col >= 0`
Returns `True` if the specified context is any cell, `False` if not. *row* and *column* are integers specifying a context.	
`IS_COLUMNLABEL(row, column)`	`IS_LABEL(row, col) \` `&& row == XRTTBL_LABEL`
Returns `True` if the specified context is a column label, `False` if not. *row* and *column* are integers specifying a context.	

`IS_LABEL(row, column)`	`row == XRTTBL_LABEL ? col >= 0 \`
	`: (col == XRTTBL_LABEL ? row >= 0 \`
	`: False)`

Returns `True` if the specified context is any label, `False` if not. *row* and *column* are integers specifying a context.	
`IS_ROWLABEL(row, column)`	`IS_LABEL(row, col) \` `&& col == XRTTBL_LABEL`
Returns `True` if the specified context is a row label, `False` if not. *row* and *column* are integers specifying a context.	

Macro Name	Definition

`LABEL_VALUE(labels, pos)`	`LABEL_VALUE_EXISTS(labels, pos) \`
	`? labels->list[pos] \`
	`: NULL`

Returns an XmString containing a label value, or `NULL` if there is no value for the label. *labels* is an `XrtTblXmStringList` row or column label structure; *pos* is an integer specifying the row (for row labels) or column (for column labels).	
`LABEL_VALUE_EXISTS(labels, pos)`	`labels && pos >= 0 \` `&& pos < labels->length`
Returns `True` if the label contains a value, `False` if it does not. *labels* is an `XrtTblXmStringList` row or column label structure; *pos* is an integer specifying the row (for row labels) or column (for column labels).	

D

Translations and Actions Reference

Default Translations
Action Routines

D.1 Default Translations

XRT/table inherits translations from XmManager, and defines the following translations of its own:

XRT/table Widget Translations

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Btn1Down>	ResizeCell(Start) TraverseToCell(Pointer) Select(Start)
Shift ~Alt ~Ctrl ~Meta <Btn1Down>	Select(Extend)
Ctrl ~Alt ~Meta ~Shift <Btn1Down>	Select(Add)
~Alt ~Meta <Btn1Motion>	ResizeCell(Move) Select(Extend)
<Key> osfPageDown	Scroll(PageDown)
<Key> osfPageUp	Scroll(PageUp)
~Alt ~Ctrl ~Meta ~Shift <Btn1Up>	Select(End) ResizeCell(End)
~Alt ~Meta <Btn1Up>	Select(End) ResizeCell(End)
Shift <Btn2Down>	Drag(Move)
Ctrl <Btn2Down>	Drag(Copy)
<Btn2Down>	Drag(Move, Move, Copy)

Default XmText Translations Installed by XRT/table

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Btn1Down>	ResizeCell(Start) Select(Start) grab-focus()[a]
Ctrl ~Alt ~Meta ~Shift <Btn1Down>	move-destination()[a] Select(Add)
~Alt ~Ctrl ~Meta ~Shift <Btn1Motion>	ResizeCell(Move) Select(Extend) extend-adjust()[a]
<Key> osfPageDown	Scroll(PageDown)
<Key> osfPageUp	Scroll(PageUp)
Ctrl ~Alt ~Meta <Key> osfDown	TraverseToCell(Down) Select(Current)
Shift ~Alt ~Ctrl ~Meta <Key> Tab	TraverseToCell(Left) Select(Current)
Ctrl ~Alt ~Meta ~Shift <Key> osfLeft	TraverseToCell(Left) Select(Current)
~Alt ~Ctrl ~Meta <Key> Tab	TraverseToCell(Right) Select(Current)
Ctrl ~Alt ~Meta <Key> osfRight	TraverseToCell(Right) Select(Current)
Ctrl ~Alt ~Meta <Key> osfUp	TraverseToCell(Up) Select(Current)
~Alt ~Ctrl ~Meta ~Shift <Btn1Up>	Select(End) ResizeCell(End) extend-end()[a]
<Key> osfCancel	CancelEdit(False) Select(Cancel)
<Key> osfActivate	CommitEdit(False)
Ctrl ~Alt ~Meta <Key> Tab	TraverseNext()
Ctrl Shift ~Alt ~Meta <Key> Tab	TraversePrev()

a. This is an XmText action.

The XmText (current cell) and XmScrollbar (table scrollbars) widgets used by XRT/table have their default translations installed.

The following translations are defined for widgets in cells/labels::

Widgets Contained in Cells

Event	Action or Actions
Shift ~Ctrl ~Meta ~Alt <Key>Tab	TraverseToCell(Left) Select(Current)
~Shift Ctrl ~Meta ~Alt <Key>osfLeft	TraverseToCell(Left) Select(Current)
~Ctrl ~Meta ~Alt <Key>Tab	TraverseToCell(Right) Select(Current)
Ctrl ~Meta ~Alt <Key>osfRight	TraverseToCell(Right) Select(Current)
Ctrl ~Meta ~Alt <Key>osfUp	TraverseToCell(Up) Select(Current)
Ctrl ~Meta ~Alt <Key>osfDown	TraverseToCell(Down) Select(Current)
Shift Ctrl ~Meta ~Alt <Key>Tab	TraversePrev()
Ctrl ~Meta ~Alt <Key>Tab	TraverseNext()

D.2 Action Routines

XRT/table defines the following action routines:

CancelEdit()

Discards the current cell's edited cell value and restores the original cell value from the table's `XrtTblCellValues` structure by calling the *XrtTblCancelEdit()* method. See page 193 for complete information on *XrtTblCancelEdit()*. This action takes one parameter that specifies whether to unmap the XmText widget:

`True`	Unmap XmText widget
`False`	Do not unmap XmText widget

CommitEdit()

Attempts to write an edited cell to the table's `XrtTblCellValues` structure by calling the *XrtTblCommitEdit()* method. *XrtTblCommitEdit()* triggers any ValidateCell callbacks to validate the edited cell value. See page 194 for complete information on *XrtTblCommitEdit()*. This action takes one parameter that specifies whether to unmap the XmText widget if successful:

`True`	Unmap XmText widget
`False`	Do not unmap XmText widget

Drag()

Performs a drag and drop operation. This action takes either one parameter or three. Each parameter is one of the following:

`Copy`	Copy dragged data
`Move`	Move dragged data (destroying it at drag site)
`NoAction`	Suppress drag and drop operation

When only one parameter is specified, it indicates whether to move or copy all of the dragged data. When three parameters are specified, they control the drag and drop operation as follows:

- The first parameter specifies whether entire rows being dragged are to be copied or moved.
- The second parameter specifies whether entire columns being dragged are to be copied or moved.
- The third parameter specifies whether ranges of cells being dragged are to be copied or moved.

NoAction()

Does nothing, triggering no callbacks. By default, this action routine is not bound to any event. Setting an event to call this routine effectively disables the translation. For example, to turn off keyboard scrolling, set new translations for <Key> osfPageDown and <Key> osfPageUp that call *NoAction()*.

ResizeCell()

Resizes a row and/or column. *ResizeCell()* handles each stage of interactive resizing–starting a resize, selecting the new size, and ending a resize. This action triggers any callbacks specified by `XmNxrtTblResizeCellCallback`, passing each callback a pointer to an `XrtTblResizeCellCallbackStruct` structure. Interactive resizing changes the value of the `XmNxrtTblPixelHeight` and/or `XmNxrtTblPixelWidth` resources. This action takes one parameter that specifies the interactive resize stage:

`Start`	Begin interactive resize; triggers ResizeCell callback with reason set to `XRTBL_REASON_RESIZE_BEGIN`
`Move`	Select new height/width; updates dashed lines
`End`	Finish resize selection; triggers ResizeCell callback with reason set to `XRTBL_REASON_RESIZE_END`

Scroll()

Scrolls the table. *Scroll()* calculates new values for the `XmNxrtTblLeftColumn` and `XmNxrtTblTopRow` resources and applies them to the widget. This action takes one parameter that specifies the scroll direction:

`PageLeft`	Scroll left one page
`PageRight`	Scroll right one page
`PageUp`	Scroll up one page
`PageDown`	Scroll down one page

Select()

Selects one or more range of cells. *Select()* handles each stage of interactive selection–starting a selection, extending the selection, and ending the selection. This action triggers any callbacks specified by `XmNxrtTblSelectCallback`, passing each callback a pointer to an `XrtTblSelectCallbackStruct` structure. Interactive selection changes the value of the `XmNxrtTblSelectedCellList` resource. This action takes one parameter that determines how the cell is selected (no selection is possible on a label):

`Start`	Select cell if `XmNxrtTblSelectionPolicy` is not `XRTTBL_SELECT_NONE`
`Current`	Select current cell if `XmNxrtTblSelectionPolicy` is not `XRTTBL_SELECT_NONE`
`Extend`	Extend selected region to include cell if `XmNxrtTblSelectionPolicy` is `XRTTBL_SELECT_RANGE` or `XRTTBL_SELECT_MULTIRANGE`
`Add`	Select cell if `XmNxrtTblSelectionPolicy` is `XRTTBL_SELECT_MULTIRANGE`

`Cancel`	Cancels the current selection
`End`	Finish a selection

TraverseNext()

Traverses out of the table to the next widget in the tab group.

TraversePrev()

Traverses out of the table to the previous widget in the tab group.

TraverseToCell()

Traverses the current cell from one location to another, scrolling to make it visible if necessary. *TraverseToCell()* handles each stage of interactive traversal–validating edited cell, determining traversal destination, and entering the cell. This action first calls the *CommitEdit()* action. If successful, it then determines the traversal destination and triggers any callbacks specified by `XmNxrtTblTraverseCellCallback`, passing each a pointer to an `XrtTblTraverseCellCallbackStruct` structure. It then scrolls to display the new cell, and triggers any callbacks specified by `XmNxrtTblEnterCellCallback` passing each a pointer to an `XrtTblEnterCellCallbackStruct` structure, before entering the cell. *TraverseToCell()* takes one parameter that specifies the traversal direction:

`Pointer`	Traverse to cell user clicked on
`Left`	Traverse left to first traversable cell
`Right`	Traverse right to first traversable cell
`Up`	Traverse up to first traversable cell
`Down`	Traverse down to first traversable cell

E

Data Types

This appendix lists the XRT/table data types in alphabetical order. The C language definition of structures is also provided.

XrtTblAction

Enumeration used to specify action with `XmNxrtTblContextValueCallback`:

```
XRTTBL_ACTION_CALCULATE
XRTTBL_ACTION_DRAW
```

XrtTblAlignment

Enumeration used to specify cell and label text alignment with `XmNxrtTblAlignmentContext` and `XmNxrtTblAlignmentSeries`:

```
XRTTBL_ALIGNMENT_TOPBEGIN
XRTTBL_ALIGNMENT_TOPCENTER
XRTTBL_ALIGNMENT_TOPEND
XRTTBL_ALIGNMENT_MIDDLEBEGIN
XRTTBL_ALIGNMENT_MIDDLECENTER
XRTTBL_ALIGNMENT_MIDDLEEND
XRTTBL_ALIGNMENT_BOTTOMBEGIN
XRTTBL_ALIGNMENT_BOTTOMCENTER
XRTTBL_ALIGNMENT_BOTTOMEND
```

XrtTblAnyValue

A union of all `XrtTblValue`-type structures:

```
typedef union {
    XrtTblArgListValue      arg;
    XrtTblBooleanValue      b;
    XrtTblColorValue        color;
    XrtTblFontListValue     f;
    XrtTblIntValue          i;
    XrtTblPickListValue     pl;
    XrtTblPixmapValue       pm;
    XrtTblPointerValue      p;
    XrtTblStringValue       s;
    XrtTblWidgetValue       w;
} XrtTblAnyValue;
```

XrtTblArgListValue

A member of a series specifying ArgList values:

```
typedef struct {
    XrtTblContext       context;
    ArgList             value;
    int                 num_args;
} XrtTblArgListValue;
```

XrtTblAttachment

Enumeration used to specify attachment of the ends of table scrollbars with `XmNxrtTblHorizScrollBarLeftAttachment`, `XmNxrtTblHorizScrollBarRightAttachment`, `XmNxrtTblVertScrollBarTopAttachment` and `XmNxrtTblVertScrollBarBottomAttachment`:

```
XRTTBL_ATTACH_CELLS
XRTTBL_ATTACH_SIDE
```

XrtTblBooleanValue

A member of a series specifying Boolean values:

```
typedef struct {
    XrtTblContext       context;
    Boolean             value;
} XrtTblBooleanValue;
```

XrtTblBorderType

Enumeration used to specify cell and label border types with `XmNxrtTblBorderTypeContext` and `XmNxrtTblBorderTypeSeries`:

```
XRTTBL_BORDER_NONE
XRTTBL_BORDER_ETCHED_IN
XRTTBL_BORDER_ETCHED_OUT
XRTTBL_BORDER_FRAME_IN
XRTTBL_BORDER_FRAME_OUT
XRTTBL_BORDER_IN
XRTTBL_BORDER_OUT
XRTTBL_BORDER_PLAIN
```

XrtTblBorderSides

Enumeration used to specify cell/label sides bordered with `XmNxrtTblBorderSidesContext` and `XmNxrtTblBorderSidesSeries`:

```
XRTTBL_BORDERSIDE_NONE
XRTTBL_BORDERSIDE_ALL
XRTTBL_BORDERSIDE_LEFT
XRTTBL_BORDERSIDE_RIGHT
XRTTBL_BORDERSIDE_TOP
XRTTBL_BORDERSIDE_BOTTOM
```

XrtTblCase

Enumeration used to specify the case conversion of user cell editing with `XmNxrtTblCaseContext` and `XmNxrtTblCaseSeries`:

```
XRTTBL_CASE_UPPER
XRTTBL_CASE_LOWER
XRTTBL_CASE_AS_IS
```

XrtTblCellValueCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblCellValueCallback` list:

```
typedef struct {
    XrtTblReason    reason;
    XEvent          *event;
    int             row, column;
    XmString        string;
    Boolean         store;
    Boolean         free_string;
} XrtTblCellValueCallbackStruct;
```

XrtTblCellValues

Structure used to contain cell values. Accessed using `CELL_VALUE` macro:

```
typedef struct {
    int                     rows;
    XrtTblXmStringSet       set_mode;
    XrtTblXmStringList      **values;
} XrtTblCellValues;
```

XrtTblColorValue

A member of a series specifying color values:

```
typedef struct {
    XrtTblContext       context;
    struct {
        String          name;
        Pixel           pixel;
        Pixel           top_shadow;
        Pixel           bottom_shadow;
    } value;
} XrtTblColorValue;
```

XrtTblColumnPlacement

Enumeration used to specify the placement of vertically-oriented areas of the table with `XmNxrtTblFrozenColumnPlacement`, `XmNxrtTblRowLabelPlacement`, and `XmNxrtTblVertScrollBarPlacement`:

```
XRTTBL_PLACE_LEFT
XRTTBL_PLACE_RIGHT
```

XrtTblCompareStruct

Structure used by the *XrtTblSortTable()* method:

```
typedef struct {
    Widget          table;
    int             row, column;
    String          string;
} XrtTblCompareStruct;
```

XrtTblContext

Structure used to define the location of a cell or label. It can also specify a row or column (possibly including the label), or even the whole table. See section 2.5 on page 26 for more information:

```
typedef struct {
    int         row, column;
} XrtTblContext;
```

XrtTblContextReason

Enumeration used to specify the context reason with `XmNxrtTblContextValueCallback`:

```
XRTTBL_CONTEXT_ALIGNMENT,
XRTTBL_CONTEXT_ARGLIST,
XRTTBL_CONTEXT_BACKGROUND,
XRTTBL_CONTEXT_BORDERSIDES,
XRTTBL_CONTEXT_BORDERTYPE,
XRTTBL_CONTEXT_BOTTOMSHADOW,
XRTTBL_CONTEXT_CASE,
XRTTBL_CONTEXT_CHAR_HEIGHT,
XRTTBL_CONTEXT_CHAR_WIDTH,
XRTTBL_CONTEXT_CLIP_PIXMAP,
XRTTBL_CONTEXT_CURRENCYPICKLIST,
XRTTBL_CONTEXT_DATATYPE,
XRTTBL_CONTEXT_DATEPICKLIST,
XRTTBL_CONTEXT_EDITABLE,
XRTTBL_CONTEXT_FIELD_ARGLIST,
XRTTBL_CONTEXT_FLOATPICKLIST,
XRTTBL_CONTEXT_FOREGROUND,
XRTTBL_CONTEXT_INTPICKLIST,
XRTTBL_CONTEXT_MAXLENGTH,
XRTTBL_CONTEXT_MULTILINE,
XRTTBL_CONTEXT_PIXEL_HEIGHT,
XRTTBL_CONTEXT_PIXEL_WIDTH,
XRTTBL_CONTEXT_PIXMAP,
XRTTBL_CONTEXT_PIXMAP_LAYOUT,
XRTTBL_CONTEXT_PIXMAP_SHADOW,
XRTTBL_CONTEXT_PS_FONTLIST,
XRTTBL_CONTEXT_SENSITIVE,
XRTTBL_CONTEXT_STRINGPICKLIST,
XRTTBL_CONTEXT_TOP_SHADOW,
XRTTBL_CONTEXT_USERDATA,
XRTTBL_CONTEXT_WIDGET_SHADOW,
XRTTBL_CONTEXT_WORDWRAP
```

XrtTblContextValue

Union containing context value information passed to callbacks on the `XmNxrtTblContextValueCallback` list:

```
typedef union {
    XrtTblAlignment     alignment;
    ArgList             arglist;
    Pixel               background;
    XrtTblBorderSides   border_sides;
    XrtTblBorderType    border_type;
    Pixel               bottom_shadow;
    int                 char_height;
    int                 char_width;
    Pixmap              clip_pixmap;
    XrtTblDataType      datatype;
    Boolean             editable;
    ArgList             field_arglist;
    Pixel               foreground;
    int                 maxlength;
    Boolean             multiline;
    int                 pixel_height;
    int                 pixel_width;
    Pixmap              pixmap;
    XrtTblPixmapLayout  pixmap_layout;
    Boolean             pixmap_shadow;
    String              ps_fontlist;
    Boolean             sensitive;
    XrtTblCase          string_case;
    Pixel               top_shadow;
    XtPointer           userdata;
    Boolean             widget_shadow;
    Boolean             wordwrap;
} XrtTblContextValue;
```

XrtTblContextValueCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblContextValueCallback` list:

```
typedef struct {
    int                 reason;
    int                 row, column;
    XEvent              *event;
    XrtTblAction        action;
    XrtTblContextReason context_reason;
    XrtTblContextValue value;
} XrtTblContextValueCallbackStruct;
```

XrtTblCreateWidgetCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblCreateWidgetCallback` list:

```
typedef struct {
    XrtTblReason    reason;
    XEvent          *event;
    int             row, column;
    Widget          parent;
    WidgetClass     widget_class;
    String          class_name, name;
    Widget          clone_source;
    Widget          new_widget;
} XrtTblCreateWidgetCallbackStruct;
```

XrtTblDataType

Enumeration used to specify the cell data type with `XmNxrtTblDataTypeContext` and `XmNxrtTblDataTypeSeries`:

```
XRTTBL_TYPE_FIELD_CURRENCY = 0,     /* XrtCurrencyField */
XRTTBL_TYPE_FIELD_DATE,             /* XrtDateField */
XRTTBL_TYPE_FIELD_FLOAT,            /* XrtFloatField */
XRTTBL_TYPE_FIELD_INT,              /* XrtIntField */
XRTTBL_TYPE_FIELD_NONE,
XRTTBL_TYPE_FIELD_STRING,           /* XrtStringField */
XRTTBL_TYPE_FIELD_COMBO_CURRENCY,   /* XrtComboBox with ... */
XRTTBL_TYPE_FIELD_COMBO_DATE,
XRTTBL_TYPE_FIELD_COMBO_FLOAT,
XRTTBL_TYPE_FIELD_COMBO_INT,
XRTTBL_TYPE_FIELD_COMBO_STRING,
XRTTBL_TYPE_FIELD_SPIN_CURRENCY,    /* XrtSpinBox with ... */
XRTTBL_TYPE_FIELD_SPIN_DATE,
XRTTBL_TYPE_FIELD_SPIN_FLOAT,
XRTTBL_TYPE_FIELD_SPIN_INT,
XRTTBL_TYPE_FIELD_SPIN_STRING,
XRTTBL_TYPE_BOOLEAN,                /* Native table types */
XRTTBL_TYPE_DOUBLE,
XRTTBL_TYPE_FLOAT,
XRTTBL_TYPE_INT,
XRTTBL_TYPE_STRING,
XRTTBL_TYPE_LAST = 100              /* User defined after this */
```

XrtTblDataTypeRoutine

Structure that defines the validation method for a particular data type with `XmNxrtTblDataTypeList`:

```
typedef struct {
    XrtTblDataType      type;
    XtCallbackProc      routine;
} XrtTblDataTypeRoutine;
```

XrtTblDisplayWidgetCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblDisplayWidgetCallback` list:

```
typedef struct {
    XrtTblReason    reason;
    XEvent          *event;
    int             row, column;
    Widget          clone_source;
    Widget          widget;
} XrtTblDisplayWidgetCallbackStruct;
```

XrtTblDispScrollBar

Enumeration used to specify when to display scrollbars with `XmNxrtTblDisplayHorizScrollBar` and `XmNxrtTblDisplayVertScrollBar`:

```
XRTTBL_DISPSB_AS_NEEDED
XRTTBL_DISPSB_ALWAYS
XRTTBL_DISPSB_NEVER
```

XrtTblDoubleBuffer

Enumeration used to control double-buffering with `XmNxrtTblDoubleBuffer`:

```
XRTTBL_DOUBLEBUFFER_NONE
XRTTBL_DOUBLEBUFFER_CELL
XRTTBL_DOUBLEBUFFER_WINDOW
```

XrtTblDrawPSarg

Enumeration used to specify arguments with *XrtTblDrawPS()* and *XrtTblVaDrawPS()*:

```
XRTTBL_PS_ALL
XRTTBL_PS_AS_IS
XRTTBL_PS_AUTO
XRTTBL_PS_BEST
XRTTBL_PS_CELL_RANGE
XRTTBL_PS_CM
XRTTBL_PS_COLOR
XRTTBL_PS_COL_PAGEBREAKS
XRTTBL_PS_EDGE_PAGES
XRTTBL_PS_GRID_TYPE
XRTTBL_PS_INCHES
XRTTBL_PS_LANDSCAPE
XRTTBL_PS_LINE
XRTTBL_PS_MARGIN_BOTTOM
XRTTBL_PS_MARGIN_LEFT
XRTTBL_PS_MARGIN_RIGHT
XRTTBL_PS_MARGIN_TOP
XRTTBL_PS_MONO
XRTTBL_PS_NONE
XRTTBL_PS_ORIENTATION
XRTTBL_PS_PAPERSIZE_HEIGHT
XRTTBL_PS_PAPERSIZE_WIDTH
XRTTBL_PS_PORTRAIT
XRTTBL_PS_ROW_PAGEBREAKS
XRTTBL_PS_SCALE
XRTTBL_PS_SHADOW
XRTTBL_PS_SHOW_COL_LABELS
XRTTBL_PS_SHOW_FROZEN_CELLS
XRTTBL_PS_SHOWPAGE
XRTTBL_PS_SHOW_ROW_LABELS
XRTTBL_PS_UNITS
XRTTBL_PS_SHOW_FROZEN_ROWS
XRTTBL_PS_SHOW_FROZEN_COLS
XRTTBL_PS_SHOW_PIXMAPS
XRTTBL_PS_SHOW_TITLES
XRTTBL_PS_SHOW_WIDGETS
XRTTBL_PS_NUM_COPIES
XRTTBL_PS_FOOTER
XRTTBL_PS_FOOTER_FONT
XRTTBL_PS_FOOTER_MARGIN
XRTTBL_PS_FOOTER_POINT_SIZE
XRTTBL_PS_HEADER
XRTTBL_PS_HEADER_FONT
XRTTBL_PS_HEADER_MARGIN
XRTTBL_PS_HEADER_POINT_SIZE
```

XrtTblEnterCellCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblEnterCellCallback` list:

```
typedef struct {
    XrtTblReason    reason;
    XEvent          *event;
    String          *params;
    Cardinal        num_params;
    int             row, column;
    Boolean         doit;
} XrtTblEnterCellCallbackStruct;
```

XrtTblExportCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblExportCallback` list:

```
typedef struct {
    XrtTblReason    reason;
    XEvent          *event;
    int             row, column;
    String          target;
    Boolean         internal;
    XrtTblRange     range;
    unsigned char   operation;
    Boolean         doit;
    Boolean         bell;
} XrtTblExportCallbackStruct;
```

XrtTblExposeCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblExposeCallback` list:

```
typedef struct {
    XrtTblReason    reason;
    XEvent          *event;
    XrtTblRange     range;
} XrtTblExposeCallbackStruct;
```

XrtTblFontListValue

A member of a series specifying font list values:

```
typedef struct {
    XrtTblContext       context;
    struct {
        XmFontList      fontList;
        Dimension       width, height;
        XmStringCharSet *tags;
        XmFontListEntry *entries;
        int             *num_entries;
    } value;
} XrtTblFontListValue;
```

XrtTblImportCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblImportCallback` list:

```
typedef struct {
    XrtTblReason    reason;
    XEvent          *event;
    int             row, column;
    String          target;
    Boolean         internal;
    unsigned char   operation;
    unsigned char   dropSiteStatus;
    Boolean         insert;
    Boolean         doit;
    Boolean         bell;
} XrtTblImportCallbackStruct;
```

XrtTblImportExportStruct

Structure specifying a drag and drop target and converter with `XmNxrtTblExportTargets` and `XmNxrtTblImportTargets`:

```
typedef struct {
    String                          target;
    XrtTblImportExportConvertProc   converter;
} XrtTblImportExportStruct;
```

XrtTblIntValue

A member of a series specifying integer values:

```
typedef struct {
    XrtTblContext       context;
    int                 value;
} XrtTblIntValue;
```

XrtTblJumpScroll

Enumeration used to specify direction of jump scrolling with `XmNxrtTblJumpScroll`:

```
XRTTBL_JUMP_NONE
XRTTBL_JUMP_HORIZONTAL
XRTTBL_JUMP_VERTICAL
XRTTBL_JUMP_ALL
```

XrtTblLabelCreateCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblLabelCreateCallback` list:

```
typedef struct {
    XrtTblReason    reason;
    XEvent          *event;
    int             row, column;
    XmString        string;
    Boolean         store;
    Boolean         free_string;
} XrtTblLabelCreateCallbackStruct;
```

XrtTblLocation

Enumeration used to specify the location of the title with `XmNxrtTblLocation`:

```
XRTTBL_LOCATION_TOP
XRTTBL_LOCATION_LEFT
XRTTBL_LOCATION_BOTTOM
XRTTBL_LOCATION_RIGHT
XRTTBL_LOCATION_NONE
```

XrtTblMode

Enumeration used to specify the display mode with `XmNxrtTblMode`:

```
XRTTBL_MODE_LIST
XRTTBL_MODE_TABLE
```

XrtTblOverflow

Enumeration used to specify cell overflowing with `XmNxrtTblOverflowContext` and `XmNxrtTblOverflowSeries`:

```
XRTTBL_OVERFLOW_NONE
XRTTBL_OVERFLOW_HORIZONTAL
```

XrtTblPickListValue

A member of a series specifying PickList values.

```
typedef struct {
    XrtTblContext       context;
    union {
        struct tm **d;
        double  *f;
        double  *c;
        int     *i;
        String  *s;
    } value;
    XrtTblDataType  type;
} XrtTblPickListValue;
```

XrtTblPixmapLayout

Enumeration used to specify the position of cell pixmaps with `XmNxrtTblPixmapLayoutContext` and `XmNxrtTblPixmapLayoutSeries`:

```
XRTTBL_PIXMAP_TOPLEFT
XRTTBL_PIXMAP_TOPCENTER
XRTTBL_PIXMAP_TOPRIGHT
XRTTBL_PIXMAP_MIDDLELEFT
XRTTBL_PIXMAP_MIDDLECENTER
XRTTBL_PIXMAP_MIDDLERIGHT
XRTTBL_PIXMAP_BOTTOMLEFT
XRTTBL_PIXMAP_BOTTOMCENTER
XRTTBL_PIXMAP_BOTTOMRIGHT
XRTTBL_PIXMAP_TILE
```

XrtTblPixmapValue

A member of a series specifying pixmap values.

```
typedef struct {
    XrtTblContext       context;
    int                 width, height;
    int                 depth;
    Pixmap              value;
} XrtTblPixmapValue;
```

XrtTblPointerValue

A member of a series specifying pointer values:

```
typedef struct {
    XrtTblContext       context;
    XtArgList           value;
} XrtTblPointerValue;
```

XrtTblPreviewScroll

Enumeration used to define when preview scrolling should be used:

```
XRTTBL_PREVIEW_NONE
XRTTBL_PREVIEW_VERTICAL
XRTTBL_PREVIEW_HORIZONTAL
XRTTBL_PREVIEW_BOTH
```

XrtTblPrintCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblPrintCallback` list:

```
typedef struct {
    int             reason;
    XEvent          *event;
    XrtTblRange     **range;
    int             current_page;
    int             total_pages;
    int             across_pages;
    int             down_pages;
    int             page_number;
    String          header, footer;
    FILE            *fp;
    Boolean         write_file_header;
    Boolean         write_file_trailer;
    Boolean         doit;
} XrtTblPrintCallbackStruct;
```

XrtTblPrintWidgetCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblPrintWidgetCallback` list:

```
typedef struct {
    int         reason;
    XEvent      *event;
    Widget      widget;
    int         row, column;
    FILE        *fp;
    int         current_page;
    int         page_number;
    Boolean     doit;
} XrtTblPrintWidgetCallbackStruct;
```

XrtTblPSarg

Structure used to define arguments for *XrtTblDrawPS()*.

```
typedef struct {
    XrtTblDrawPSarg     name;
    XtArgVal            value;
} XrtTblPSarg;
```

XrtTblPSFontMap

Structure used to define a PostScript to X font correspondence with `XmNxrtTblPSFontMapList`:

```
typedef struct {
    String      font_name, tag;
    String      PS_font;
    double      point_size;
} XrtTblPSFontMap;
```

XrtTblRange

Structure used to define a range of cells:

```
typedef struct {
    int     start_row, start_column;
    int     end_row, end_column;
} XrtTblRange;
```

XrtTblReason

Enumeration used to define the reason for a callback:

```
XRTTBL_REASON_CELLVALUE
XRTTBL_REASON_CONTEXTVALUE
XRTTBL_REASON_CREATE_WIDGET_BEGIN
XRTTBL_REASON_CREATE_WIDGET_END
XRTTBL_REASON_CURSOR_EDGE
XRTTBL_REASON_CURSOR_INSIDE
XRTTBL_REASON_CURSOR_LEAVE
XRTTBL_REASON_DISPLAY_WIDGET
XRTTBL_REASON_ENTER_CELL_BEGIN
XRTTBL_REASON_ENTER_CELL_END
XRTTBL_REASON_EXPORT_DRAG_BEGIN
XRRTBL_REASON_EXPORT_TARGET
XRTTBL_REASON_EXPORT_DRAG_END
XRTTBL_REASON_EXPOSE_BEGIN
XRTTBL_REASON_EXPOSE_END
XRTTBL_REASON_IMPORT_DRAG_MOTION
XRTTBL_REASON_IMPORT_BEGIN
XRTTBL_REASON_IMPORT_TARGET
XRTTBL_REASON_IMPORT_DROP_END
XRTTBL_REASON_LABELCREATE
XRTTBL_REASON_PRINTWIDGET_BEGIN
XRTTBL_REASON_PRINTWIDGET_END
XRTTBL_REASON_PRINT_BEGIN
XRTTBL_REASON_PRINT_END
XRTTBL_REASON_RESIZE_BEGIN
XRTTBL_REASON_RESIZE_DONE
XRTTBL_REASON_RESIZE_END
XRTTBL_REASON_SCROLL_BEGIN
XRTTBL_REASON_SCROLL_END
XRTTBL_REASON_SELECT_BEGIN
XRTTBL_REASON_SELECT_END
XRTTBL_REASON_SELECT_EXTEND
XRTTBL_REASON_SETVALUE_BEGIN
XRTTBL_REASON_SETVALUE_END
XRTTBL_REASON_SORT
XRTTBL_REASON_TRAVERSE_CELL
XRTTBL_REASON_UNSELECT
XRTTBL_REASON_VALIDATE_CELL_BEGIN
XRTTBL_REASON_VALIDATE_CELL_END
```

XrtTblRepaint

Enumeration used to specify table repainting behavior with `XmNxrtTblRepaint`:

```
XRTTBL_REPAINT_ON
XRTTBL_REPAINT_OFF
XRTTBL_REPAINT_NOCALC
```

XrtTblResize

Enumeration used to specify resizing constraints with `XmNxrtTblAllowResize` and `XmNxrtTblAllowCellResize`:

```
XRTTBL_RESIZE_NONE
XRTTBL_RESIZE_HORIZONTAL
XRTTBL_RESIZE_VERTICAL
XRTTBL_RESIZE_ALL
```

XrtTblResizeCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblResizeCallback` list:

```
typedef struct {
    XrtTblReason    reason;
    XEvent          *event;
    Dimension       width, height;
    Dimension       visible_width;
    Dimension       visible_height;
} XrtTblResizeCallbackStruct;
```

XrtTblResizeCellCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblResizeCellCallback` list:

```
typedef struct {
    XrtTblReason    reason;
    XEvent          *event;
    String          *params;
    Cardinal        num_params;
    int             row, column;
    int             current_row_height,
    int             current_column_width,
    int             new_row_height,
    int             new_column_width,
    Boolean         doit;
} XrtTblResizeCellCallbackStruct;
```

XrtTblRowPlacement

Enumeration used to specify the placement of horizontally-oriented areas of the table with `XmNxrtTblFrozenRowPlacement`, `XmNxrtTblColumnLabelPlacement`, and `XmNxrtTblHorizScrollBarPlacement`:

```
XRTTBL_PLACE_TOP
XRTTBL_PLACE_BOTTOM
```

XrtTblSBPosition

Enumeration used to specify how the scrollbars are attached to the table widget with `XmNxrtTblHorizScrollBarPosition` and `XmNxrtTblVertScrollBarPosition`:

```
XRTTBL_SBPOSITION_CELLS
XRTTBL_SBPOSITION_SIDE
```

XrtTblScrollCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblScrollCallback` list:

```
typedef struct {
    int                         reason;
    XEvent                      *event;
    unsigned char               direction;
    int                         new_value;
    String                      preview_text;
    XmScrollBarCallbackStruct   *sb_cbs;
} XrtTblScrollCallbackStruct;
```

XrtTblSelectCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblSelectCallback` list:

```
typedef struct {
    XrtTblReason    reason;
    XEvent          *event;
    String          *params;
    Cardinal        num_params;
    int             range_index;
    XrtTblRange     *range;
    int             row, column;
    int             click_count;
    Boolean         doit;
} XrtTblSelectCallbackStruct;
```

XrtTblSelectionPolicy

Enumeration used to define the selection policy with `XmNxrtTblSelectionPolicy`:

```
XRTTBL_SELECT_NONE
XRTTBL_SELECT_SINGLE
XRTTBL_SELECT_RANGE
XRTTBL_SELECT_MULTIRANGE
```

XrtTblSeries

Structure used to define a list of pointers to `XrtTblAnyValue` instances:

```
typedef struct {
    XrtTblValueType     type;
    String              name;
    int                 last_index;
    int                 length;
    XrtTblAnyValue      **list;
} XrtTblSeries;
```

XrtTblSortCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblSortCallback` list:

```
typedef struct {
    int                 reason;
    XEvent              *event;
    int                 column;
    XrtTblCompareStruct *data;
    int                 num_rows;
} XrtTblSortCallbackStruct;
```

XrtTblStringValue

A member of a series specifying String values:

```
typedef struct {
    XrtTblContext       context;
    String              value;
} XrtTblStringValue;
```

XrtTblTrackCursorCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblTrackCursorCallback` list:

```
typedef struct {
    XrtTblReason        reason;
    XEvent              *event;
    int                 row, column;
    XrtTblBorderSides   sides;
    Boolean             enter_cell;
    Cursor              old_cursor;
    Cursor              new_cursor;
} XrtTblTrackCursorCallbackStruct;
```

XrtTblTraverseCellCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblTraverseCellCallback` list:

```
typedef struct {
    XrtTblReason    reason;
    XEvent          *event;
    String          *params;
    Cardinal        num_params;
    int             row, column;
    int             next_row;
    int             next_column;
} XrtTblTraverseCellCallbackStruct;
```

XrtTblValidateCellCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblValidateCellCallback` list:

```
typedef struct {
    XrtTblReason    reason;
    XEvent          *event;
    int             row, column;
    Boolean         value_changed;
    XrtTblDataType  datatype;
    XtArgVal        *converted_value;
    XtPointer       user_data;
    String          value;
    Boolean         doit;
    Boolean         bell;
} XrtTblValidateCellCallbackStruct;
```

XrtTblValidatePolicy

Enumeration used to define cell value validation policy:

```
XRTTBL_VALIDATE_USER_EDIT
XRTTBL_VALIDATE_SET
XRTTBL_VALIDATE_NEVER
XRTTBL_VALIDATE_ALWAYSXrtTblValueType
```

XrtTblValueType

Enumeration used as an argument in *XrtTblSeries()* procedures:

```
XRTTBL_VALUE_ARGLIST = 1,
XRTTBL_VALUE_BOOLEAN,
XRTTBL_VALUE_COLOR,
XRTTBL_VALUE_CURRENCY_PICKLIST,
XRTTBL_VALUE_DATE_PICKLIST,
XRTTBL_VALUE_FONTLIST,
XRTTBL_VALUE_FLOAT_PICKLIST,
XRTTBL_VALUE_INT_PICKLIST,
XRTTBL_VALUE_INT,
XRTTBL_VALUE_PIXMAP,
XRTTBL_VALUE_POINTER,
XRTTBL_VALUE_STRING,
XRTTBL_VALUE_STRING_PICKLIST,
XRTTBL_VALUE_WIDGET
XRTTBL_VALUE_ARGLIST = 1,
XRTTBL_VALUE_BOOLEAN,
XRTTBL_VALUE_COLOR,
XRTTBL_VALUE_CURRENCY_PICKLIST,
XRTTBL_VALUE_DATE_PICKLIST,
XRTTBL_VALUE_FONTLIST,
XRTTBL_VALUE_FLOAT_PICKLIST,
XRTTBL_VALUE_INT_PICKLIST,
XRTTBL_VALUE_INT,
XRTTBL_VALUE_PIXMAP,
XRTTBL_VALUE_POINTER,
XRTTBL_VALUE_STRING,
XRTTBL_VALUE_STRING_PICKLIST,
XRTTBL_VALUE_WIDGET
```

XrtTblWidgetClass

Structure that defines a widget class with `XmNxrtTblWidgetClassList`:

```
typedef struct {
    String          name;
    WidgetClass     *widget_class;
} XrtTblWidgetClass;
```

XrtTblWidgetPos

Sub-structure of `XrtTblWidgetValue` used to store widget position.

```
typedef struct {
    Widget      widget;
    int         row, column;
    Window      parent_window;
    Boolean     moved;
} XrtTblWidgetPos;
```

XrtTblWidgetSetValueCallbackStruct

Structure that defines the information passed to callbacks on the `XmNxrtTblWidgetSetValueCallback` list:

```
typedef struct {
    int         reason;
    XEvent      *event;
    int         row, column;
    Widget      widget;
    Arg         *arglist;
    int         num_arglist;
    Boolean     doit;
} XrtTblWidgetSetValueCallbackStruct;
```

XrtTblWidgetValue

A member of a series specifying widget values.

```
typedef struct {
    XrtTblContext               context;
    Window                      parent_window;
    struct {
        Widget                  widget;
        int                     width, height;
        XrtTblWidgetPos         *list;
        int                     num_list;
        String                  resource_string;
    } value;
} XrtTblWidgetValue;
```

XrtTblXmStringGet

Enumeration used to specify how XmString resources are retrieved from the table with `XmNxrtTblXmStringGetMode`:

```
XRTTBL_GET_STRING
XRTTBL_GET_XMSTRING
```

XrtTblXmStringList

Structure containing row labels, column labels, or cell values for one row:

```
typedef struct {
    int             length;
    XmString        *list;
} XrtTblXmStringList;
```

If Strings are expected by the table, then it is necessary to cast them as XmStrings before assigning them to a StringList.

XrtTblXmStringSet

Enumeration used to specify how XmString resources are retrieved from the table with `XmNxrtTblXmStringSetMode`:

```
XRTTBL_SET_EXAMINE
XRTTBL_SET_STRING
XRTTBL_SET_XMSTRING
```

XrtTblXmStringProc

A procedure prototype used by `XmNxrtTblCellValueFunc` and `XmNxrtTblLabelCreateFunc`:

```
typedef XmString (*XrtTblXmStringProc)();
```

F

Resource Types and Classes

This appendix lists the resource type and resource class for each XRT/table resource:

Name	Class	Type
XmNxrtTblAlignmentContext	XmCxrtTblAlignment	XmRxrtTblAlignment
XmNxrtTblAlignmentSeries	XmCxrtTblAlignmentSeries	XmRxrtTblAlignmentSeries
XmNxrtTblAllowCellResize	XmCxrtTblAllowCellResize	XmRxrtTblResize
XmNxrtTblAllowResize	XmCxrtTblAllowResize	XmRxrtTblResize
XmNxrtTblArgListContext	XmCxrtTblArgList	XmRxrtTblArgList
XmNxrtTblArgListSeries	XmCxrtTblArgListSeries	XmRxrtTblArgListSeries
XmNxrtTblBackgroundContext	XmCBackground	XmRString
XmNxrtTblBackgroundPixelContext	XmCBackground	XmRPixel
XmNxrtTblBackgroundSeries	XmCxrtTblColorSeries	XmRxrtTblColorSeries
XmNxrtTblBorderSidesContext	XmCxrtTblBorderSides	XmRxrtTblBorderSides
XmNxrtTblBorderSidesSeries	XmCxrtTblBorderSidesSeries	XmRxrtTblBorderSidesSeries
XmNxrtTblBorderTypeContext	XmCxrtTblBorderType	XmRxrtTblBorderType
XmNxrtTblBorderTypeSeries	XmCxrtTblBorderTypeSeries	XmRxrtTblBorderTypeSeries
XmNxrtTblBottomShadowContext	XmCBackground	XmRPixel
XmNxrtTblCaseContext	XmCxrtTblCase	XmRxrtTblCase
XmNxrtTblCaseSeries	XmCxrtTblCaseSeries	XmRxrtTblCaseSeries
XmNxrtTblCellValueCallback	XmCCallback	XmRCallback
XmNxrtTblCellValueContext	XmCxrtTblCellValues	XmRXmString
XmNxrtTblCellValueFunc	XmCCallback	XtRFunction

Name	Class	Type
XmNxrtTblCellValues	XmCxrtTblCellValues	XmRxrtTblCellValues
XmNxrtTblCharHeightContext	XmCxrtTblHeight	XmRxrtTblCellSize
XmNxrtTblCharHeightSeries	XmCxrtTblSizeSeries	XmRxrtTblCellSizeSeries
XmNxrtTblCharWidthContext	XmCxrtTblWidth	XmRxrtTblCellSize
XmNxrtTblCharWidthSeries	XmCxrtTblSizeSeries	XmRxrtTblCellSizeSeries
XmNxrtTblClipChildren	XmCReadOnly	XmRWidgetList
XmNxrtTblClipChildrenPersistent	XmCBoolean	XmRBoolean
XmNxrtTblClipPixmapContext	XmCPixmap	XmRxrtTblClipPixmap
XmNxrtTblClipPixmapSeries	XmCxrtTblPixmapSeries	XmRxrtTblClipPixmapSeries
XmNxrtTblColumnLabelOffset	XmCxrtTblOffset	XmRInt
XmNxrtTblColumnLabelPlacement	XmCxrtTblRowPlacement	XmRxrtTblRowPlacement
XmNxrtTblColumnLabels	XmCxrtTblXmStringList	XmRxrtTblXmStringList
XmNxrtTblContext	XmCxrtTblContext	XmRxrtTblContext
XmNxrtTblContextValueCallback	XmCCallback	XmRCallback
XmNxrtTblCreateWidgetCallback	XmCCallback	XmRCallback
XmNxrtTblCurrencyPickListContext	XmCxrtTblCurrencyPickList	XmRxrtTblCurrencyPickList
XmNxrtTblCurrencyPickListSeries	XmCxrtTblCurrencyPickListSeries	XmRxrtTblCurrencyPickListSeries
XmNxrtTblCursorCorner	XmCCursor	XmRCursor
XmNxrtTblCursorHorizontal	XmCCursor	XmRCursor
XmNxrtTblCursorNontraversable	XmCCursor	XmRCursor
XmNxrtTblCursorTraversable	XmCCursor	XmRCursor
XmNxrtTblCursorVertical	XmCCursor	XmRCursor
XmNxrtTblDataTypeContext	XmCxrtTblDataType	XmRxrtTblDataType
XmNxrtTblDataTypeList	XmCxrtTblDataTypeList	XmRxrtTblDataTypeList
XmNxrtTblDataTypeSeries	XmCxrtTblDataTypeSeries	XmRxrtTblDataTypeSeries
XmNxrtTblDatePickListContext	XmCxrtTblDatePickList	XmRxrtTblDatePickList
XmNxrtTblDatePickListSeries	XmCxrtTblDatePickListSeries	XmRxrtTblDatePickListSeries
XmNxrtTblDebug	XmCxrtTblDebug	XmRBoolean
XmNxrtTblDisplayClipArrows	XmCxrtTblDisplayClipArrows	XmRBoolean
XmNxrtTblDisplayHorizScrollBar	XmCxrtTblDispScrollBar	XmRxrtTblDispScrollBar
XmNxrtTblDisplayVertScrollBar	XmCxrtTblDispScrollBar	XmRxrtTblDispScrollBar
XmNxrtTblDisplayWidgetCallback	XmCCallback	XmRCallback

Name	Class	Type
XmNxrtTblDoubleBuffer	XmCxrtTblDoubleBuffer	XmRxrtTblDoubleBuffer
XmNxrtTblEditableContext	XmCxrtTblEditable	XmRBoolean
XmNxrtTblEditableSeries	XmCxrtTblEditabilitySeries	XmRxrtTblBooleanSeries
XmNxrtTblEnterCellCallback	XmCCallback	XmRCallback
XmNxrtTblExportCallback	XmCCallback	XmRCallback
XmNxrtTblExportDelimiter	XmCxrtTblDelimiter	XmRString
XmNxrtTblExportTargets	XmCxrtTblTargets	XmRxrtTblTargets
XmNxrtTblExposeCallback	XmCCallback	XmRCallback
XmNxrtTblFieldArgListContext	XmCxrtTblArgList	XmRxrtTblArgList
XmNxrtTblFieldArgListSeries	XmCxrtTblFieldArgListSeries	XmRxrtTblArgListSeries
XmNxrtTblFloatPickListContext	XmCxrtTblFloatPickList	XmRxrtTblFloatPickList
XmNxrtTblFloatPickListSeries	XmCxrtTblFloatPickListSeries	XmRxrtTblFloatPickListSeries
XmNxrtTblFontListContext	XmCFontList	XmRFontList
XmNxrtTblFontListSeries	XmCxrtTblFontListSeries	XmRxrtTblFontListSeries
XmNxrtTblForegroundContext	XmCForeground	XmRString
XmNxrtTblForegroundPixelContext	XmCForeground	XmRPixel
XmNxrtTblForegroundSeries	XmCxrtTblColorSeries	XmRxrtTblColorSeries
XmNxrtTblFrameBorderType	XmCxrtTblBorderType	XmRxrtTblBorderType
XmNxrtTblFrameShadowThickness	XmCShadowThickness	XmRDimension
XmNxrtTblFrozenColumnPlacement	XmCxrtTblColumnPlacement	XmRxrtTblColumnPlacement
XmNxrtTblFrozenColumns	XmCxrtTblFrozenColumns	XmRInt
XmNxrtTblFrozenRowPlacement	XmCxrtTblRowPlacement	XmRxrtTblRowPlacement
XmNxrtTblFrozenRows	XmCxrtTblFrozenRows	XmRInt
XmNxrtTblHorizontalScrollBar	XmCHorizontalScrollBar	XmRWidget
XmNxrtTblHorizScrollBarLeftAttachment	XmCxrtTblScrollBarAttachment	XmRxrtTblAttachment
XmNxrtTblHorizScrollBarPlacement	XmCxrtTblRowPlacement	XmRxrtTblRowPlacement
XmNxrtTblHorizScrollBarPosition	XmCxrtTblScrollBarPosition	XmRxrtTblScrollBarPosition
XmNxrtTblHorizScrollBarRightAttachment	XmCxrtTblScrollBarAttachment	XmRxrtTblAttachment
XmNxrtTblImportCallback	XmCCallback	XmRCallback
XmNxrtTblImportDelimiter	XmCxrtTblDelimiter	XmRString
XmNxrtTblImportTargets	XmCxrtTblTargets	XmRxrtTblTargets
XmNxrtTblIntPickListContext	XmCxrtTblIntPickList	XmRxrtTblIntPickList

Name	Class	Type
XmNxrtTblIntPickListSeries	XmCxrtTblIntPickListSeries	XmRxrtTblIntPickListSeries
XmNxrtTblJumpScroll	XmCxrtTblJumpScroll	XmRxrtTblJumpScroll
XmNxrtTblLabelContext	XmCLabel	XmRXmString
XmNxrtTblLabelCreateCallback	XmCCallback	XmRCallback
XmNxrtTblLabelCreateFunc	XmCCallback	XtRFunction
XmNxrtTblLeftColumn	XmCxrtTblLeftColumn	XmRInt
XmNxrtTblMarginHeight	XmCMarginHeight	XmRDimension
XmNxrtTblMarginWidth	XmCMarginWidth	XmRDimension
XmNxrtTblMaxLengthContext	XmCMaxLength	XmRxrtTblInt
XmNxrtTblMaxLengthSeries	XmCxrtTblMaxLengthSeries	XmRxrtTblIntSeries
XmNxrtTblMinCellVisibility	XmCxrtTblMinCellVisibility	XmRInt
XmNxrtTblMinFlexibleHeight	XmCxrtTblMinFlexibleHeight	XmRDimension
XmNxrtTblMinFlexibleWidth	XmCxrtTblMinFlexibleWidth	XmRDimension
XmNxrtTblMode	XmCxrtTblMode	XmRxrtTblMode
XmNxrtTblMultilineContext	XmCxrtTblMultiline	XmRBoolean
XmNxrtTblMultilineSeries	XmCxrtTblEditabilitySeries	XmRxrtTblBooleanSeries
XmNxrtTblNumColumns	XmCxrtTblNumColumns	XmRInt
XmNxrtTblNumRows	XmCxrtTblNumRows	XmRInt
XmNxrtTblOverflowContext	XmCxrtTblOverflow	XmRxrtTblOverflow
XmNxrtTblOverflowSeries	XmCxrtTblOverflowSeries	XmRxrtTblOverflowSeries
XmNxrtTblPixelHeightContext	XmCxrtTblHeight	XmRxrtTblCellSize
XmNxrtTblPixelHeightSeries	XmCxrtTblSizeSeries	XmRxrtTblCellSizeSeries
XmNxrtTblPixelWidthContext	XmCxrtTblWidth	XmRxrtTblCellSize
XmNxrtTblPixelWidthSeries	XmCxrtTblSizeSeries	XmRxrtTblCellSizeSeries
XmNxrtTblPixmapContext	XmCPixmap	XmRPixmap
XmNxrtTblPixmapLayoutContext	XmCxrtTblPixmapLayout	XmRxrtTblPixmapLayout
XmNxrtTblPixmapLayoutSeries	XmCxrtTblPixmapLayoutSeries	XmRxrtTblPixmapLayoutSeries
XmNxrtTblPixmapSeries	XmCxrtTblPixmapSeries	XmRxrtTblPixmapSeries
XmNxrtTblPixmapShadowContext	XmCxrtTblPixmapShadow	XmRBoolean
XmNxrtTblPixmapShadowSeries	XmCxrtTblPixmapShadowSeries	XmRxrtTblBooleanSeries
XmNxrtTblPreviewScrolling	XmCxrtTblPreviewScrolling	XmRxrtTblPreviewScrolling
XmNxrtTblPrintCallback	XmCCallback	XmRCallback

Name	Class	Type
XmNxrtTblPrintWidgetCallback	XmCCallback	XmRCallback
XmNxrtTblPSFontListContext	XmCString	XmRString
XmNxrtTblPSFontListSeries	XmCxrtTblStringSeries	XmRxrtTblStringSeries
XmNxrtTblPSFontMapList	XmCxrtTblPSFontMapList	XmRxrtTblPSFontMapList
XmNxrtTblRangeContext	XmCxrtTblRangeContext	XmRxrtTblRange
XmNxrtTblRepaint	XmCxrtTblRepaint	XmRxrtTblRepaint
XmNxrtTblRepeatBackgroundColors	XmCxrtTblStringTable	XmRStringTable
XmNxrtTblRepeatForegroundColors	XmCxrtTblStringTable	XmRStringTable
XmNxrtTblResizeCallback	XmCCallback	XmRCallback
XmNxrtTblResizeCellCallback	XmCCallback	XmRCallback
XmNxrtTblRowLabelOffset	XmCxrtTblOffset	XmRInt
XmNxrtTblRowLabelPlacement	XmCxrtTblColumnPlacement	XmRxrtTblColumnPlacement
XmNxrtTblRowLabels	XmCxrtTblXmStringList	XmRxrtTblXmStringList
XmNxrtTblScrollCallback	XmCCallback	XmRCallback
XmNxrtTblSelectCallback	XmCCallback	XmRCallback
XmNxrtTblSelectedBackground	XmCxrtTblSelectedBackground	XmRString
XmNxrtTblSelectedCellList	XmCxrtTblRangeList	XmRxrtTblRangeList
XmNxrtTblSelectedForeground	XmCxrtTblSelectedForeground	XmRString
XmNxrtTblSelectionPolicy	XmCSelectionPolicy	XmRxrtTblSelectionPolicy
XmNxrtTblSensitiveContext	XmCxrtTblSensitive	XmRBoolean
XmNxrtTblSensitiveSeries	XmCxrtTblEditabilitySeries	XmRxrtTblBooleanSeries
XmNxrtTblSetTextCursorPos	XmCxrtTblSetTextCursorPos	XmRBoolean
XmNxrtTblShadowThickness	XmCShadowThickness	XmRDimension
XmNxrtTblSortCallback	XmCCallback	XmRCallback
XmNxrtTblSpace	XmCSpace	XmRDimension
XmNxrtTblSpanList	XmCxrtTblRangeList	XmRxrtTblRangeList
XmNxrtTblStringPickListContext	XmCxrtTblStringPickList	XmRxrtTblStringPickList
XmNxrtTblStringPickListSeries	XmCxrtTblStringPickListSeries	XmRxrtTblStringPickListSeries
XmNxrtTblText	XmCxrtTblText	XmRWidget
XmNxrtTblTextInitSelect	XmCxrtTblTextInitSelect	XmRBoolean
XmNxrtTblTopRow	XmCxrtTblTopRow	XmRInt
XmNxrtTblTopShadowContext	XmCBackground	XmRPixel

Name	Class	Type
XmNxrtTblTrackCursor	XmCxrtTblTrackCursor	XmRBoolean
XmNxrtTblTrackCursorCallback	XmCCallback	XmRCallback
XmNxrtTblTraversableContext	XmCxrtTblTraversable	XmRBoolean
XmNxrtTblTraversableSeries	XmCxrtTblEditabilitySeries	XmRxrtTblBooleanSeries
XmNxrtTblTraverseCellCallback	XmCCallback	XmRCallback
XmNxrtTblUserDataContext	XmCUserData	XmRPointer
XmNxrtTblUserDataSeries	XmCUserData	XmRxrtTblPointerSeries
XmNxrtTblValidateCellCallback	XmCCallback	XmRCallback
XmNxrtTblValidatePolicy	XmCxrtTblValidatePolicy	XmRxrtTblValidatePolicy
XmNxrtTblVerticalScrollBar	XmCVerticalScrollBar	XmRWidget
XmNxrtTblVertScrollBarBottomAttachment	XmCxrtTblScrollBarAttachment	XmRxrtTblAttachment
XmNxrtTblVertScrollBarPlacement	XmCxrtTblColumnPlacement	XmRxrtTblColumnPlacement
XmNxrtTblVertScrollBarPosition	XmCxrtTblScrollBarPosition	XmRxrtTblScrollBarPosition
XmNxrtTblVertScrollBarTopAttachment	XmCxrtTblScrollBarAttachment	XmRxrtTblAttachment
XmNxrtTblVisibleColumns	XmCxrtTblVisibleColumns	XmRInt
XmNxrtTblVisibleRows	XmCxrtTblVisibleRows	XmRInt
XmNxrtTblWidgetClassList	XmCxrtTblWidgetClassList	XmRxrtTblWidgetClassList
XmNxrtTblWidgetContext	XmCxrtTblWidget	XmRxrtTblWidget
XmNxrtTblWidgetSeries	XmCxrtTblWidgetSeries	XmRxrtTblWidgetSeries
XmNxrtTblWidgetSetValueCallback	XmCCallback	XmRCallback
XmNxrtTblWidgetShadowContext	XmCxrtTblWidgetShadow	XmRBoolean
XmNxrtTblWidgetShadowSeries	XmCxrtTblWidgetShadowSeries	XmRxrtTblBooleanSeries
XmNxrtTblWordWrapContext	XmCxrtTblWordWrap	XmRBoolean
XmNxrtTblWordWrapSeries	XmCxrtTblWordWrapSeries	XmRxrtTblBooleanSeries
XmNxrtTblXmStringGetMode	XmCxrtTblXmStringGetMode	XmRxrtTblXmStringGetMode
XmNxrtTblXmStringSetMode	XmCxrtTblXmStringSetMode	XmRxrtTblXmStringSetMode

Constraint Resources		
Name	**Class**	**Type**
XmNxrtTblBottomOffset	XmCxrtTblOffset	XmRxrtTblInt
XmNxrtTblLeftOffset	XmCxrtTblOffset	XmRxrtTblInt
XmNxrtTblLocation	XmCxrtTblLocation	XmRxrtTblLocation
XmNxrtTblResizeToCell	XmCBoolean	XmRBoolean
XmNxrtTblRightOffset	XmCxrtTblOffset	XmRxrtTblInt
XmNxrtTblTopOffset	XmCxrtTblOffset	XmRxrtTblInt
XmNxrtTblWidgetLocation	XmCxrtTblContext	XmRxrtTblContext

G

Resource File and GUI Builder Usage

Syntax
Example Resource File

This appendix lists the syntax required for resource strings in resource files and GUI builder tools so that they are recognized by the XRT/table resource converters. It also lists a sample resource file. XRT/table only defines resource converters for the data types not already defined by Xt and Motif.

All table resources can be set in a resource file except resources that use a procedure to set their value. Use Series resources to specify attributes over portions of the table.

G.1 Syntax

Comments
Any text after an exclamation mark (`!`) on a line is ignored.

Continuation
Lines are continued on subsequent lines in the file by making a backslash (`\`) the last character in the line.

Booleans
"True" or "False" in upper, lower, or mixed case.

Enumerated Types
All resources with enumerated types use a consistent naming convention based on the enumerated values. For all resource types, the leading `XRTTBL_` is dropped. The resource values can be entered in upper, lower, or mixed case. For example, each of the following resource specifications causes the value `XRTTBL_BORDER_ETCHED_OUT` to be used for `XmNxrtTblBorderType`:

```
*.XtXrtTable.xrtTblBorderType: BORDER_ETCHED_OUT
*.XtXrtTable.xrtTblBorderType: border_etched_out
*.XtXrtTable.xrtTblBorderType: BorDer_Etched_out
```

Series Resources

Series resource values are specified as a list of space-delimited series members, each surrounded by parentheses. The first two items in a member specify the context; the third item is the resource value. Members can be separated by spaces. To specify a left or right parenthesis as part of a value, it must be preceded by two backslashes (\\(). The row and column index can be one of the following:

`<integer>`	a row or column number
`label`	a row or column label
`all`	all rows or columns, including labels
`allcells`	all cells in the row or column
`<integer> - <integer>`	arange of rows or columns
`<integer>, <integer>`	start/end location of a range of cells/labels

The following example shows some valid series members:

```
*.XtXrtTable.xrtTblForegroundSeries: (all all red) \
    (all label blue) (2 3 blue) (1-3 1 green) \
    (5,5 7,7 green)
```

Picklist Series Resources

Picklist series resources uses the same format as other series resources, as described above in Series Resources. Specifically, the first two entries are the row/column combination. All entries in the picklist are separated by commas.

The following are some examples of picklist series resources:

```
*.XtXrtTable.xrtTblIntPickListSeries: \
    (0 0-2 1,3,5,7,9,11) \
    (0 3-5 2,4,6,8,10,12,14,16,18,20,24,26) \
    (2 ALLCELLS 2,4,6,8,10,12,14,16,18,20,24,26)
```

The above assigns the integer picklist *1,3,5,7,9,11* to columns 0, 1 and 2 in row 0; the integer picklist *2,4,6,8,10,12,14,16,18,20,24,26* to columns 3, 4 and 5 in row 0; and the integer picklist *2,4,6,8,10,12,14,16,18,20,24,26* to all the columns in row 2, excluding the row label column.

Note that integer picklists support binary (0b) and hexidecimal (0x) formats, as well as the standard decimal format.

```
*.XtXrtTable.xrtTblStringPickListSeries: \
    (1 ALLCELLS John,Paul,George,Ringo) \
    (4 ALLCELLS Here are,some more,resources,from a,resource,file) \
    (6 0-2 These,values,"have been"\\,"loaded",from a,resource,file)
```

The above assigns the string picklist *John,Paul,George,Ringo* to all cells in row 1; the string picklist *Here are,some more,resources,from a,resource,file* to all cells in row 4 and *These,values,"have been"\\,"loaded",from a,resource,file* to columns 0,1 and 2 in row 6.

Note the embedded double quotes and escaped comma in the last picklist.

```
*XtXrtTable.xrtTblCurrencyPickListSeries: \
    (3 ALLCELLS 0,10.0,11.0,12.0,13.0-12.231,+45.5543e3)

*XtXrtTable.xrtTblFloatPickListSeries: \
    (5 ALLCELLS 10.0,11.0,12.0,13.0,-100,2e5,+45.5543e3)
```

The above examples create float and currency picklists. The syntax and formats are identical for both.

Note that Float and Currency picklists support standard floating point and exponential notation.

```
*XtXrtTable.xrtTblDatePickListSeries: \
    (7 ALLCELLS NOW, TODAY, 1999-01-01 01:01:01, \
     1999-Apr-01 01:01:01, 01:01:01, NOW)
```

The above example creates a date picklist for all the cells in row 7. The date picklist supports the following formats:

```
YYYY-M-D h:m:s
YYYY-MMM-D h:m:s
YYYY-M-D
YYYY-MMM-D
h:m:s
TODAY
NOW
```

Note that `TODAY` is midnight of the current date and `NOW` is the current date and time.

Cell Values

`XmNxrtTblCellValues` can be specified either directly or through a file. If a filename is specified, the resource converter opens the file and calls *XrtTblReadAscii()*. The specification can include an environment variable or a character that indicates the column delimiter used in the file. An example:

```
*.XtXrtTable.xrtTblCellValues: $MYDIR/values.dat |
```

To include cell values directly in the resource file, this resource is specified as a comma-separated list of values, one list for each row. Each list is surrounded by parentheses. Rows do not need to have an identical number of column values. To specify a comma or parenthesis as part of the value, precede it with two backslashes (`\\`). To omit a value, enter two commas with no intervening space (`,,`). To specify a multiple-line value, insert a newline (`\n`) as appropriate. An example:

```
*.XtXrtTable.xrtTblCellValues: (a,b,c) \
    (e,f\\,and g) \
    (Our number:\n\\(416\\) 594-1026) \
```

Labels

Resources of type `XmRxrtTblXmStringList` are specified as comma-separated values. Spaces are assumed to be part of the value. To specify a comma as part of a value, it must be preceded by two backslashes (`\\,`). To omit a value, enter two commas with no intervening space (`,,`). To specify a multiple-line value, insert a newline (`\n`) as appropriate. An example:

```
*.XtXrtTable.xrtTblRowLabels: first,second, \
    and\\,a third,fourth,,sixth
```

Cell/Label Pixmaps

`XmNxrtTblPixmapSeries` is specified through a file. The resource converter opens the file and calls procedures to read the file. The pixmap must be in either XBM or

XPM format. The specification can include an environment variable. To specify that no pixmap appear in a context, use NONE or UNSPECIFIED_PIXMAP. An example:

```
*.XtXrtTable.xrtTblPixmapSeries: \
    (label all none) (allcells 0 $MYDIR/myimage.xpm) \
    (all label xlogo32)
```

Cell/Label Widget

XmNxrtTblWidgetSeries is specified with a widget class and an optional widget name. The resource values can be entered in upper, lower, or mixed case. To specify that no widget appear in a context, use NONE. See section 6.1.2 on page 115 for complete information. An example:

```
*.XtXrtTable.xrtTblWidgetSeries: \
    (label all XmLabel) (1 3 XmPushbutton.my_button) \
    (all label none) (0 0 XtXrtTable.inner_table) \
```

Border Sides

XmNxrtTblBorderSidesSeries is specified by OR-ing together one or more of the valid XrtTblBorderSides enumerated values with a plus sign (+), with no intervening spaces, as shown below:

```
*.XtXrtTable.xrtTblBorderSidesSeries: \
    (label all borderside_top+borderside_bottom) \
    (allcells allcells borderside_all)
```

Span List and Selected Cell List

Resources of type XmRxrtTblRangeList are specified as a list of ranges, with each range surrounded by parentheses. The first item in a range specification is the first and last row; the second item is the first and last column; for example, (0-1 2-5). Another way to specify a range is with the locations of the top-left cell and bottom-right cell; for example, (0,0 3,3). Spaces may appear between the members of the range list. Use ALL or ALLCELLS to reference all cells in a row or column. An example:

```
*.XtXrtTable.xrtTblSelectedCellList: (1-5 6-9) \
    (10,13 12,20) (ALL 21)
*.XtXrtTable.xrtTblSpanList: (0,0 1,1) (allcells 25)
```

Mouse Pointers

Resources of type XmRCursor are specified with the symbol name of the cursor, omitting the XC_ prefix. An example:

```
*.XtXrtTable.xrtTblCursorNontraversable: left_ptr
```

Background and Shadow Colors

Each series member of XmNxrtTblBackgroundSeries is specified with a context, the background color, and, optionally, the top shadow and bottom shadow colors. When the top or bottom shadow is omitted, XRT/table calculates it. The top shadow color is specified before the bottom, separated by commas.

The following example sets the background to "white", top shadow to "grey90", and bottom shadow to "grey95":

```
*.XtXrtTable.xrtTblBackgroundSeries: \
    (1 3 white,grey90,grey95)
```

PostScript Font Map Lists

XmNxrtTblPSFontMapList is specified as a comma-separated list of values, one list for each font definition. Each list is surrounded by parentheses. For example:

```
*.XtXrtTable.xrtTblPSFontMapList: \
    (-*-helvetica-medium-r-*-*-*-*-*-*-*-iso8859-1,,\
    Helvetica-Narrow, 0)\
    (-*-helvetica-bold-r-*-*-*-*-*-*-*-iso8859-1,,\
    Helvetica-Narrow-Bold, 0)\
    (-*-helvetica-medium-o-*-*-*-*-*-*-*-iso8859-1,,\
    Helvetica-Narrow-Oblique, 0)\
    (-*-helvetica-bold-o-*-*-*-*-*-*-*-iso8859-1,,\
    Helvetica-Narrow-BoldOblique, 0)
```

Repeated Colors

XmNxrtTblRepeatBackgroundColors and XmNxrtTblRepeatForegroundColors are specified as comma-separated lists of colors:

```
*.XtXrtTable.xrtTblRepeatBackgroundColors: \
    red,white,blue
```

Setting the XmString Converter

The *XrtTblSetXmStringConverter()* procedure sets the XmRXmString resource converter to create XmString values for cells and labels. This procedure is passed no arguments and returns no value, and must be called before any table is created

The XmRXmString resource converter uses the following syntax to specify embedded font changes:

@font@	font's tag in XmNfontList
@P@	return to previous font
\@	a literal "@"

For example, suppose the following string is specified in a resource file:

```
For a @bold@free evaluation@P@ of any product
```

If the bold font is defined in the font list as

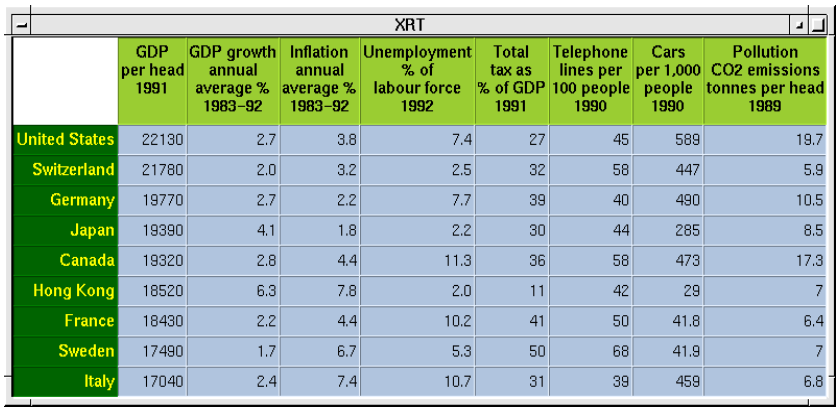
```
fixed,-*-times-bold-r-normal-*-140-*=bold
```

the converted XmString value is displayed as

For a **free evaluation** of any product

G.2 Example Resource File

The following resource file defines the table shown below:



	GDP per head 1991	GDP growth annual average % 1983–92	Inflation annual average % 1983–92	Unemployment % of labour force 1992	Total tax as % of GDP 1991	Telephone lines per 100 people 1990	Cars per 1,000 people 1990	Pollution CO2 emissions tonnes per head 1989
United States	22130	2.7	3.8	7.4	27	45	589	19.7
Switzerland	21780	2.0	3.2	2.5	32	58	447	5.9
Germany	19770	2.7	2.2	7.7	39	40	490	10.5
Japan	19390	4.1	1.8	2.2	30	44	285	8.5
Canada	19320	2.8	4.4	11.3	36	58	473	17.3
Hong Kong	18520	6.3	7.8	2.0	11	42	29	7
France	18430	2.2	4.4	10.2	41	50	41.8	6.4
Sweden	17490	1.7	6.7	5.3	50	68	41.9	7
Italy	17040	2.4	7.4	10.7	31	39	459	6.8

```
! Sample resource file similar to $XRTHOME/src/table/samples/econ.res
*xrtTblAlignmentSeries:             (ALL ALL ALIGNMENT_TOPEND)\
(LABEL ALL ALIGNMENT_TOPCENTER)
*xrtTblAllowCellResize:             RESIZE_ALL
*xrtTblAllowResize:                 RESIZE_NONE
*xrtTblBackgroundSeries:            (ALL ALL tan)(LABEL ALL Yellow Green)\
(ALL LABEL Dark Green)(ALLCELLS     ALLCELLS Light Steel Blue)
*xrtTblBackingStore:                False
*xrtTblBorderTypeSeries:            (ALL ALL BORDER_IN)
*xrtTblCaseSeries:                  (ALL ALL CASE_AS_IS)
*xrtTblCellValues:                  $XRTHOME/src/table/data/economics.ascii
*xrtTblCharHeightSeries:            (ALL ALL 1)(LABEL ALL 4)
*xrtTblCharWidthSeries:             (ALL ALL 10)
*xrtTblClipChildrenPersistent:      False
*xrtTblColumnLabelOffset:           0
*xrtTblColumnLabelPlacement:        PLACE_TOP
*xrtTblColumnLabels:                GDP\\nper head\\n1991,\
GDP growth\\nannual\\naverage %\\n1983-92,\
Inflation\\nannual\\naverage %\\n1983-92,\
Unemployment\\n% of\\nlabour force\\n1992,\
Total\\ntax as\\n% of GDP\\n1991,\
Telephone\\nlines per\\n100 people\\n1990,\
Cars\\nper 1\\,000\\npeople\\n1990,\
Pollution\\nCO2 emissions\\ntonnes per head\\n1989
*xrtTblDataTypeSeries:              (ALL ALL TYPE_STRING)
*xrtTblDisplayClipArrows:           True
*xrtTblDisplayHorizScrollBar:       DISPSB_AS_NEEDED
*xrtTblDisplayVertScrollBar:        DISPSB_AS_NEEDED
*xrtTblDoubleBuffer:                DOUBLEBUFFER_CELL
*xrtTblEditableSeries:              (ALL ALL True)
*xrtTblExportDelimiter:
*xrtTblFontListSeries:\
(ALL ALL -Adobe-Helvetica-Medium-R-Normal--14-140-75-75-P-77-ISO8859-1)\
(LABEL ALL -Adobe-Helvetica-Bold-R-Normal--14-140-75-75-P-82-ISO8859-1)\
(ALL LABEL -Adobe-Helvetica-Bold-R-Normal--14-140-75-75-P-82-ISO8859-1)
*xrtTblForegroundSeries:                  (ALL ALL black)(ALL LABEL Yellow)
```

```
*xrtTblFrameBorderType:                  BORDER_NONE
*xrtTblFrameShadowThickness:             0
*xrtTblFrozenColumnPlacement:            PLACE_LEFT
*xrtTblFrozenColumns:                    0
*xrtTblFrozenRowPlacement:               PLACE_TOP
*xrtTblFrozenRows:                       0
*xrtTblHorizScrollBarLeftAttachment:     ATTACH_CELLS
*xrtTblHorizScrollBarPlacement:          PLACE_BOTTOM
*xrtTblHorizScrollBarPosition:           SBPOSITION_CELLS
*xrtTblHorizScrollBarRightAttachment:    ATTACH_CELLS
*xrtTblImportDelimiter:
*xrtTblJumpScroll:                       JUMP_NONE
*xrtTblLeftColumn:                       0
*xrtTblMarginHeight:                     4
*xrtTblMarginWidth:                      2
*xrtTblMaxLengthSeries:                  (ALL ALL MAXINT)
*xrtTblMinCellVisibility:                100
*xrtTblMinFlexibleHeight:                5
*xrtTblMinFlexibleWidth:                 5
*xrtTblMode:MODE_TABLE
*xrtTblMultilineSeries:                  (ALL ALL False)
*xrtTblNumColumns:                       8
*xrtTblNumRows:                          9
*xrtTblOverflowSeries:                   (ALL ALL OVERFLOW_NONE)
*xrtTblPSFontListSeries:                 (ALL ALL null)
*xrtTblPixelHeightSeries:                (ALL ALL VARIABLE)
*xrtTblPixelWidthSeries:                 (ALL ALL VARIABLE)
*xrtTblRepeatBackgroundColors:           Inherited
*xrtTblRepeatForegroundColors:           Inherited
*xrtTblRowLabelOffset:                   0
*xrtTblRowLabelPlacement:                PLACE_LEFT
*xrtTblRowLabels:                        United States,Switzerland,Germany,Japan,\
Canada,Hong Kong,France,Sweden,Italy
*xrtTblSelectedBackground:               Black
*xrtTblSelectedForeground:               White
*xrtTblSelectionPolicy:                  SELECT_NONE
*xrtTblSetTextCursorPos:                 False
*xrtTblShadowThickness:                  1
*xrtTblSpace:                            6
*xrtTblTextInitSelect:                   False
*xrtTblTopRow:                           0
*xrtTblTrackCursor:                      True
*xrtTblTraversableSeries:                (ALL ALL False)
*xrtTblValidatePolicy:                   VALIDATE_USER_EDIT
*xrtTblVertScrollBarBottomAttachment:    ATTACH_CELLS
*xrtTblVertScrollBarPlacement:           PLACE_RIGHT
*xrtTblVertScrollBarPosition:            SBPOSITION_CELLS
*xrtTblVertScrollBarTopAttachment:       ATTACH_CELLS
*xrtTblVisibleColumns:                   8
*xrtTblVisibleRows:                      9
*xrtTblWidgetSeries:                     (ALL ALL NONE)
*xrtTblWidgetShadowSeries:               (ALL ALL False)
*xrtTblWordWrapSeries:                   (ALL ALL False)
*xrtTblXmStringGetMode:                  GET_XMSTRING
*xrtTblXmStringSetMode:                  SET_EXAMINE
```

H

UIL Arguments

This appendix provides an alphabetical listing of XRT/table's UIL arguments and their data types. A second table provides XRT/table UIL reasons. For more information on using XRT/table with UIL, see section 6.14 on page 142.

Note that all arguments whose names end in "Context" are applied to the entire table. Also note that it is not possible to specify any of the PickList resources in a UIL file, since UIL only supports data types used by standard Motif widgets.

UIL Argument Name	Argument Type
XmNxrtTblAlignmentContext	integer
XmNxrtTblAllowCellResize	integer
XmNxrtTblAllowResize	integer
XmNxrtTblArgListContext	string
XmNxrtTblBackgroundContext	string
XmNxrtTblBackgroundPixelContext	color
XmNxrtTblBorderSidesContext	integer
XmNxrtTblBorderTypeContext	integer
XmNxrtTblBottomOffset	integer
XmNxrtTblBottomShadowContext	color
XmNxrtTblCaseContext	integer
XmNxrtTblCellValueContext	compound_string
XmNxrtTblCharHeightContext	integer
XmNxrtTblCharWidthContext	integer
XmNxrtTblClipChildrenPersistent	boolean

UIL Argument Name	Argument Type
XmNxrtTblClipPixmapContext	icon
XmNxrtTblColumnLabelOffset	integer
XmNxrtTblColumnLabelPlacement	integer
XmNxrtTblDataTypeContext	integer
XmNxrtTblDebug	boolean
XmNxrtTblDisplayClipArrows	boolean
XmNxrtTblDisplayHorizScrollBar	integer
XmNxrtTblDisplayVertScrollBar	integer
XmNxrtTblDoubleBuffer	integer
XmNxrtTblEditableContext	boolean
XmNxrtTblExportDelimiter	string
XmNxrtTblFieldArgListContext	string
XmNxrtTblFontListContext	font_table
XmNxrtTblForegroundContext	string
XmNxrtTblFrameBorderType	integer
XmNxrtTblFrameShadowThickness	integer
XmNxrtTblFrozenColumnPlacement	integer
XmNxrtTblFrozenColumns	integer
XmNxrtTblFrozenRowPlacement	integer
XmNxrtTblFrozenRows	integer
XmNxrtTblHorizScrollBarLeftAtt	integer
XmNxrtTblHorizScrollBarPlace	integer
XmNxrtTblHorizScrollBarPosition	integer
XmNxrtTblHorizScrollBarRightAtt	integer
XmNxrtTblImportDelimiter	string
XmNxrtTblJumpScroll	integer
XmNxrtTblLeftColumn	integer
XmNxrtTblLeftOffset	integer
XmNxrtTblLocation	integer
XmNxrtTblMarginHeight	integer
XmNxrtTblMarginWidth	integer
XmNxrtTblMaxLengthContext	integer

UIL Argument Name	Argument Type
XmNxrtTblMinCellVisibility	integer
XmNxrtTblMinFlexibleHeight	integer
XmNxrtTblMinFlexibleWidth	integer
XmNxrtTblMode	integer
XmNxrtTblMultilineContext	boolean
XmNxrtTblNumColumns	integer
XmNxrtTblNumRows	integer
XmNxrtTblOverflowContext	integer
XmNxrtTblPixelHeightContext	integer
XmNxrtTblPixelWidthContext	integer
XmNxrtTblPixmapContext	icon
XmNxrtTblPixmapLayoutContext	integer
XmNxrtTblPixmapShadowContext	boolean
XmNxrtTblPreviewScrolling	integer
XmNxrtTblPSFontListContext	string
XmNxrtTblRepaint	integer
XmNxrtTblRepeatBackgroundColors	asciz_string_table
XmNxrtTblRepeatForegroundColors	asciz_string_table
XmNxrtTblResizeToCell	boolean
XmNxrtTblRightOffset	integer
XmNxrtTblRowLabelOffset	integer
XmNxrtTblRowLabelPlacement	integer
XmNxrtTblSelectedBackground	string
XmNxrtTblSelectedForeground	string
XmNxrtTblSelectionPolicy	integer
XmNxrtTblSensitiveContext	boolean
XmNxrtTblSetTextCursorPos	boolean
XmNxrtTblShadowThickness	integer
XmNxrtTblSpace	integer
XmNxrtTblTextInitSelect	boolean
XmNxrtTblTopOffset	integer
XmNxrtTblTopRow	integer

UIL Argument Name	Argument Type
XmNxrtTblTopShadowContext	color
XmNxrtTblTrackCursor	boolean
XmNxrtTblTraversableContext	boolean
XmNxrtTblUseXmGetColors	boolean
XmNxrtTblValidatePolicy	integer
XmNxrtTblVertScrollBarBottomAtt	integer
XmNxrtTblVertScrollBarPlacement	integer
XmNxrtTblVertScrollBarPosition	integer
XmNxrtTblVertScrollBarTopAttach	integer
XmNxrtTblVisibleColumns	integer
XmNxrtTblVisibleRows	integer
XmNxrtTblWidgetContext	widget
XmNxrtTblWidgetShadowContext	boolean
XmNxrtTblWordWrapContext	boolean
XmNxrtTblXmStringGetMode	integer
XmNxrtTblXmStringSetMode	integer

Reasons
XmNxrtTblCellValueCallback
XmNxrtTblContextValueCallback
XmNxrtTblCreateWidgetCallback
XmNxrtTblDisplayWidgetCallback
XmNxrtTblEnterCellCallback
XmNxrtTblExportCallback
XmNxrtTblExposeCallback
XmNxrtTblImportCallback
XmNxrtTblLabelCreateCallback
XmNxrtTblPrintCallback
XmNxrtTblPrintWidgetCallback
XmNxrtTblResizeCallback
XmNxrtTblResizeCellCallback
XmNxrtTblScrollCallback

Reasons
XmNxrtTblSelectCallback
XmNxrtTblSortCallback
XmNxrtTblTrackCursorCallback
XmNxrtTblTraverseCellCallback
XmNxrtTblValidateCellCallback
XmNxrtTblWidgetSetValueCallback

I

The XrtLabel Widget

Widget Synopsis ■ *Introduction*
Using XrtLabel ■ *Use with XRT/table*
Resource Reference ■ *Resource Types and Classes*

The XrtLabel widget is a label that is similar to the Motif XmLabel widget.[1] XrtLabel is designed to add attractive titles to tables; it should not be used anywhere other than in the XRT/table widget. It has several capabilities not found in standard labels–rotation, shadow borders, and printing (when in a table).

I.1 Widget Synopsis

Include File:	*<Xm/XrtLabel.h>*
Class Name:	XtXrtLabel
Class Hierarchy:	Core → XmPrimitive → XtXrtLabel
Class Pointer:	xtXrtLabelWidgetClass
Instantiation:	`widget = XtCreateWidget(name, xtXrtLabelWidgetClass...)`
Procedures/Macros:	*XmCreateXrtLabel()*, *IsXrtLabel()*

I.2 Introduction

XrtLabel adds a new widget to the Motif toolkit. A Motif programmer manipulates this widget using the familiar Xt Intrinsics calls.

1. In the previous release, XrtLabel was referred to as "XRT/label" in this manual.

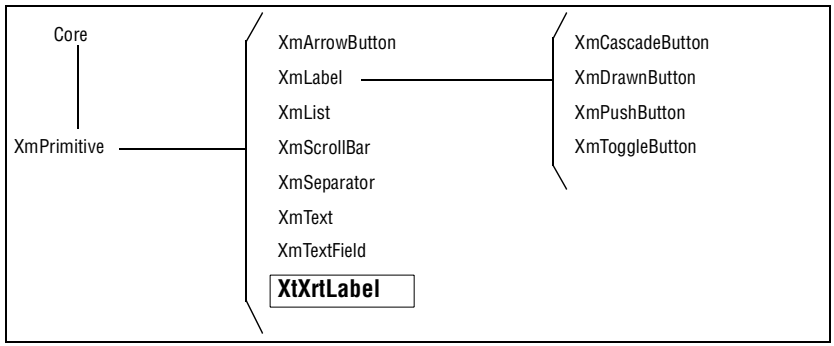


Figure 65 XrtLabel's position within the Motif Class Hierarchy

The widget has most of the resources of the standard XmLabel. The important visual components are illustrated by Figure 66.

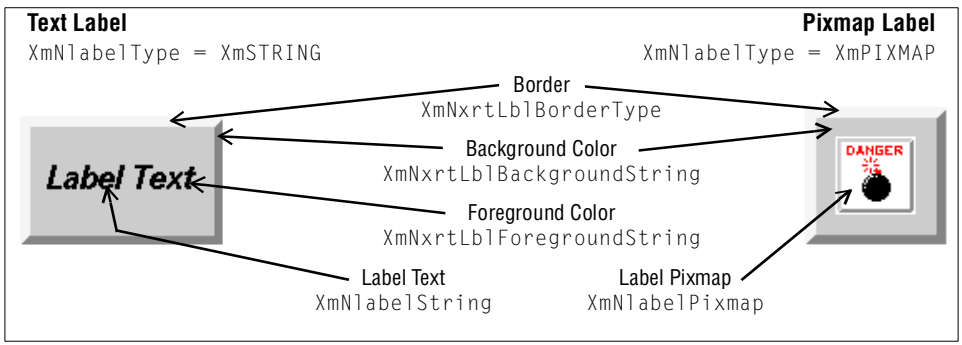


Figure 66 Visual Components of the XrtLabel widget

I.3 Using XrtLabel

Creating a Label

To create a label, include the *<Xm/XrtLabel.h>* header file and call *XtVaCreateManagedWidget()*, as shown by the following example:

```
Widget label;
label = XtVaCreateManagedWidget("label",
    xtXrtLabelWidgetClass,     table,
    ...
    NULL);
```

Label Type

XrtLabel can display either a text string or a pixmap image in the label, as shown by Figure 66. The `XmNlabelType` resource specifies the label type. To display a text label, set the label type to `XmSTRING`. To display a pixmap label, set the label type to `XmPIXMAP`.

Text labels display the value of XmNlabelString. Pixmap labels display the pixmap specified by XmNlabelPixmap.

Text Labels

The text displayed in a label can be set or changed using the XmNlabelString resource. Motif 1.2 developers can set this resource using a String or an XmString; Motif 2.x developers can only set it using XmStrings. XmNalignment controls the alignment of the label text.

The following example sets new text for a label:

```
XtVaSetValues(label,
    XmNlabelString,      "Label Text",
    XmNalignment,        XmALIGNMENT_CENTER,
    NULL);
```

Pixmap Labels

The pixmap image displayed in a label can be set or changed using the XmNlabelPixmap resource. This resource takes a pixmap as its value. XrtLabel supports pixmaps in XBM or XPM format. Pixmaps should be freed after use; XRT/table does not copy pixmaps set in cells/labels. Pixmaps appear inside the margins set for the label and XmNalignment controls the alignment of the label pixmap.

While you can store XBM/XPM images directly in your C code, it is much easier to maintain images in files and read them when needed. The easiest way to specify an image stored in a file is to use the Motif resource converter to convert the filename string to the Pixmap data type required by XmNlabelPixmap, as shown below:

```
XtVaSetValues(label,
    XmNlabelType,        XmPIXMAP,
    XtVaTypedArg,        XmNlabelPixmap,
    XmRString, "my_image.xpm",
    strlen("my_image.xpm") + 1,
    NULL);
```

To remove a pixmap from the label, set XmNlabelPixmap to XmUNSPECIFIED_PIXMAP.

Border Types

XmNxrtLblBorderType specifies the type of border drawn around the label. The border styles you can use are shown in Figure 67. You can set the width of the border using XmNshadowThickness.

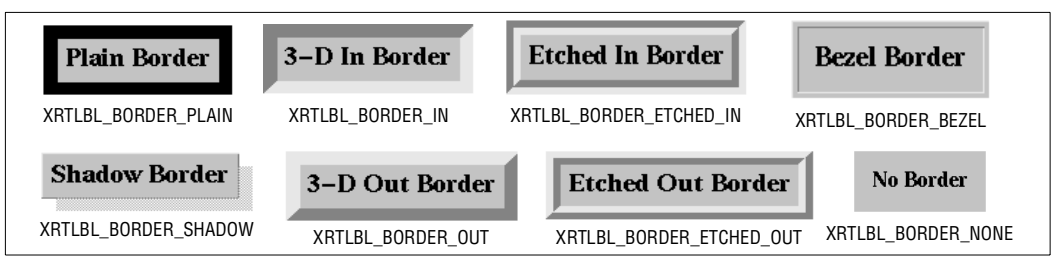


Figure 67 Border types

Rotation

The contents of a label can be rotated using the `XmNxrtLblRotation` resource. To rotate the label text or pixmap 90 degrees counter-clockwise, set this resource to `XRTLBL_ROTATE_90`. You can also rotate label contents 270 degrees counter-clockwise by setting it to `XRTLBL_ROTATE_270`.

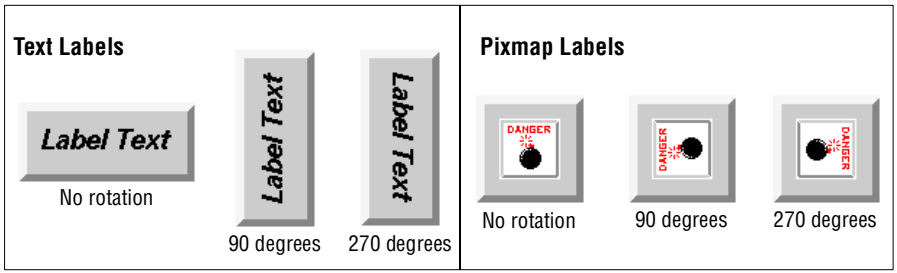


Figure 68 Label rotation

Colors

XrtLabel provides an alternate way to specify label background/foreground colors in addition to the inherited `XmNbackground` and `XmNforeground` resources. `XmNxrtLblBackgroundString` and `XmNxrtLblForegroundString` allow you to specify colors using text strings. This makes it easy to use colors from the X Server's color database (such as "LemonChiffon") or represented as hexadecimal RGB values (such as "#228B22"). See section 2.8 on page 29 for more details on how colors can be specified.

Setting one color resource automatically updates the other one. For example, setting `XmNforeground` to a Pixel value would update `XmNxrtLblForegroundString` with a RGB string representation of that value.

Setting Height and Width

By default,when an XrtLabel widget is used as a title, it is sized to the height and width of the text or pixmap it contains. To set the height and width yourself, set the `XmNrecomputeSize` resource to `False`, then use `XmNheight` and `XmNwidth` to define the size of your title.

If `XmNrecomputeSize` is set to its default value of `True`, `XmNheight` and `XmNwidth` cannot be set.

Setting other Attributes

Standard XmLabel resources inherited by XrtLabel work identically. For example, you can use `XmNfontList` to set the font list used for the label text. Appendix I.5 on page 276 lists all of the resources inherited by XrtLabel–consult your Motif programming documentation for more information on using these resources.

I.4 Use with XRT/table

XrtLabel was designed as an attractive title widget for use with XRT/table. This section briefly describes using XrtLabel with XRT/table.

Adding a Title

To add a title to a table, create the widget as a child of the table. The following example creates a label as a table title:

```
title = XtVaCreateWidget("title",
    xtXrtLabelWidgetClass,      table,
    XmNlabelString,             "A Table Title",
    XmNxrtTblLocation,          XRTTBL_LOCATION_TOP,
    NULL);
```

Positioning a Title

XRT/table's constraint resources control the location (and possibly the size) of each title. Constraint resources must be set on the label itself, not on the table. For complete information on adding titles to a table, see section 3.6 on page 46.

Cell/label Placement

XrtLabel widgets can also be placed in the cells and labels of a table. Because they can be rotated, it is useful to place them in cells you want rotated. For complete information on placing widgets in cells and labels, see section 6.1 on page 109.

The following example adds an XrtLabel widget to all column labels:

```
label = XtVaCreateWidget("label",
    xtXrtLabelWidgetClass,          table,
    XmNxrtTblWidgetLocation,
              XrtTblSetContext(XRTTBL_LABEL, XRTTBL_ALL),
    NULL);
```

Printing

When used with XRT/table as a title, XRT/table's PostScript output methods (*XrtTblDrawPS()* and *XrtTblVaDrawPS()*) will print the label using PostScript fonts and device-independent PostScript operators.

I.5 Resource Reference

This section lists all of the XrtLabel resources in alphabetical order. Listed after the resource name is its data type, default value and a list of the Xt procedures (**C**reate, **S**et, **G**et) that may be used with the resource.

XmNlabelPixmap **Pixmap** **XmUNSPECIFIED_PIXMAP** **CSG**

Overrides the inherited `XmNlabelPixmap` resource to enable it to be set in a resource file and to support pixmaps in XPM format. This resource specifies a label pixmap, displayed when `XmNlabelType` is `XmPIXMAP`.

XmNlabelString **XmString** **NULL** **CSG**

Overrides the inherited `XmNlabelString` resource to enable it to be set as a String (if linked to Motif 1.2). This resource specifies a label text value, displayed when `XmNlabelType` is `XmSTRING`. When specified as a String, the XrtLabel widget converts it internally to an XmString.

XmNshadowThickness **Dimension** **5** **CSG**

Overrides the default value (2) of this Core-inherited resource.

XmNxrtLblBackgroundString **String** **"Inherited"** **CSG**

Specifies the background color of the label. This resource is identical to the Core-inherited `XmNbackground` resource except that it takes a String for the color value. The value can be any named color from the X color database or a hexadecimal RGB color value. When set to "Inherited", the color is inherited `XmNbackground`. When set, the `XmNbackground` resource is updated as a side-effect.

On a monochrome Server, setting this resource to anything other than black causes white to be used.

XmNxrtLblBorderType **XrtLblBorderType** **XRTLBL_BORDER_IN** **CSG**

Specifies the style used for the label border. Valid values are `XRTLBL_BORDER_PLAIN`, `XRTLBL_BORDER_SHADOW`, `XRTLBL_BORDER_IN`, `XRTLBL_BORDER_OUT`, `XRTLBL_BORDER_ETCHED_IN`, `XRTLBL_BORDER_ETCHED_OUT`, and `XRTLBL_BORDER_NONE`.

The border width is specified by `XmNshadowThickness`.

XmNxrtLblForegroundString **String** **"Inherited"** **CSG**

Specifies the foreground color used for the label text. This resource is identical to the Core-inherited `XmNforeground` resource except that it takes a String for the color value. The value can be any named color from the X color database or a hexadecimal RGB color value. When set to "Inherited", the color is inherited from `XmNforeground`. When set, the `XmNforeground` resource is updated as a side-effect.

On a monochrome Server, setting this resource to anything other than white causes black to be used.

XmNxrtLblRotation **XrtLblRotation** **XRTLBL_ROTATE_NONE** **CSG**

Specifies the rotation of label contents (text or pixmap). Valid values are: `XRTLBL_ROTATE_NONE`, `XRTLBL_ROTATE_90`, or `XRTLBL_ROTATE_270`. Rotation is measured counter-clockwise.

Inherited Resources

XrtLabel inherits resources from XmLabel and its parent classes. The following lists these resources along with the parent class that defines them. Consult your Motif and Xt documentation for more information.

XmLabel Resources

XmNalignment	XmNlabelString	XmNrecomputeSize
XmNfontList	XmNlabelType	XmNstringDirection
XmNlabelInsensitivePixmap	XmNmarginHeight	
XmNlabelPixmap	XmNmarginWidth	

I.6 Resource Types and Classes

Resource	Inherited From
XmNaccelerators	Core
XmNancestorSensitive	Core
XmNbackground	Core
XmNbackgroundPixmap	Core
XmNborderColor	Core
XmNborderPixmap	Core
XmNborderWidth	Core
XmNbottomShadowColor	Primitive
XmNbottomShadowPixmap	Primitive
XmNcolormap	Core
XmNdepth	Core
XmNdestroyCallback	Core
XmNforeground	Primitive
XmNheight	Core
XmNhelpCallback	Primitive
XmNhighlightColor	Primitive
XmNhighlightOnEnter	Primitive

Resource	Inherited From
XmNhighlightPixmap	Primitive
XmNhighlightThickness	Primitive
XmNinitialResourcesPersistent	Core
XmNmappedWhenManaged	Core
XmNnavigationType	Primitive
XmNscreen	Core
XmNsensitive	Core
XmNshadowThickness	Primitive
XmNtopShadowColor	Primitive
XmNtopShadowPixmap	Primitive
XmNtranslations	Core
XmNtraversalOn	Primitive
XmNuserData	Primitive
XmNwidth	Core
XmNx	Core
XmNy	Core

Name	Class	Type
XmNlabelPixmap	XmCLabelPixmap	XmRPixmap
XmNlabelString	XmCXmString	XmRXmString
XmNshadowThickness	XmCShadowThickness	XmRHorizontalDimension
XmNxrtLblBackgroundString	XmCBackground	XmRString
XmNxrtLblBorderType	XmCxrtLblBorderType	XmRxrtLblBorderType
XmNxrtLblForegroundString	XmCForeground	XmRString
XmNxrtLblRotation	XmCxrtLblRotation	XmRxrtLblRotation

J

Sample Code

Examples
Demos
simple.c Listing

This appendix provides an overview of the sample code included with XRT/table. XRT/table provides two types of programming samples–examples, and demos. Examples are short programs that illustrate *specific* features. Demos are more complete applications that illustrate *several features* working together. Most common XRT/table programming tasks are covered in the examples and demos.

J.1 Examples

Example programs are located in *$XRTHOME/src/table/examples*:

- *simple.c* displays a very basic table that you can use to learn some XRT/table basics. It is fully discussed in Chapter 1.
- *styles* shows how to switch between three different table looks with the click of a button. It is implemented three ways: *styles_c.c* sets all resources in C code; *styles_res.c* reads all resources from a resource file (*styles.res*); and *styles_uil.c* uses UIL to set all resources (*styles_uil.uil*).
- *multiply.c* shows how to use `XmNxrtTblCellValueCallback` to specify the value of each cell on demand, an efficient way to handle very large tables.
- *list.c* shows how to mimic an XmScrolledList with a table by setting `XmNxrtTblMode`. It also shows sorting a table, and searching a table for a string.
- *entry.c* shows the use of the TraverseCell and EnterCell callbacks to control traversal and text entry into a simple form. It also shows the use of buttons and option menus within cells.
- *simple_cpp.cxx* implements *simple.c* in C++.

J.2 Demos

Demo applications are located in *$XRTHOME/src/table/demos*. There is a directory for each demo.

- *widgets*–This demo illustrates how to manage widgets in cells. One window uses XRT/table to hold a palette of "source" widgets. Another window displays an empty "destination" table. A user can drag a widget from the source palette, and drop it on a cell in the destination table.
- *form*–This demo is a data-entry form that uses two tables. One table displays an an employee list overview. When a user clicks on a cell, a second table serves as a data entry form, where a user can change employee information. It shows how to use spanned cells, widgets in cells, and pixmaps in cells.
- *play*–This demo provides direct access to all of XRT/table's resources. A user can change any resource value and immediately see how the changes affect the table. *play* saves table descriptions in resource files, which you can use in your application.
- *bombhunt*–This demo uses XRT/table in a simple game. The object is to find all of the bombs hidden under some of the squares.
- *stocks*–This demo illustrates runtime setting of colors.

Multi-Product Demos

Several demos use XRT/table with other XRT products. The following demos are located in their own directory in *$XRTHOME/src/common_demos*.

- *graph_editor*–This demo uses XRT/table and XRT/graph in one application. A table displays data that is graphed by XRT/graph. A user can edit the data in the table, and the graph updates to reflect the change.
- *mvc*–This is a demonstration of MVC that uses XRT/graph, XRT/table, and XRT/3d.
- *spreadsheet*–This demo mimics a spreadsheet application, allowing a user to load or enter data into cells. *spreadsheet* demonstrates most of XRT/table's features and perform many common table functions, including moving and deleting rows/columns, changing visual attributes (colors, borders, text alignment, etc.), changing cell editability and traversability, and printing a range of cells.
- *surface_editor*–This demo uses XRT/table and XRT/3d in one application. A table displays Z-values displayed in the 3D graph. A user can edit the Z-values and the graph updates to reflect the change.
- *table_entry*–This demo shows you how you can use XRT/field widgets with XRT/table. A simple table uses different field subclasses for validating user editing in different columns. For example, an integer field edits the Phone Number column.

J.3 simple.c Listing

This program is discussed in detail in section 1.3 on page 13. When run, the window in Figure 2 on page 13 is displayed.

```
#include <Xm/XrtTable.h>
#include <stdlib.h>

int
main(argc, argv)
int argc;
char *argv[];
{
    Widget toplevel, table, title;
    XrtTblCellValues *values;
    XrtTblXmStringList labels;
    XmString s;
    static String label_list[] =
        { "Name", "Address", "Phone Number", "Position", "Salary" };
    FILE *fp;

    /* Open data file */
    if (!(fp = fopen("simple.dat", "r"))) {
        perror("simple.dat");
        exit(0);
    }

    /* Initialize Intrinsics */
    toplevel = XtInitialize(argv[0], "Simple", NULL, 0, &argc, argv);

    /* Read data values, and load into table */
    values = XrtTblReadAscii(toplevel, fp, ',');
    labels.length = XtNumber(label_list);
    labels.list = (XmString *)label_list;
    table = XtVaCreateManagedWidget("table",
        xtXrtTableWidgetClass,       toplevel,
        XmNxrtTblNumRows,            7,
        XmNxrtTblNumColumns,         5,
        XmNxrtTblCellValues,         values,
        XmNxrtTblColumnLabels,       &labels,
        XmNxrtTblBackgroundContext,  "honeydew1",
        XmNxrtTblForegroundContext,  "blue4",
        XmNxrtTblPixelWidthContext,  XRTTBL_VARIABLE,
        XmNxrtTblPixelHeightContext, XRTTBL_VARIABLE,
        XmNxrtTblXmStringSetMode,    XRTTBL_SET_XMSTRING, /* Motif 2.x only */
        NULL);
    XrtTblDestroyCellValues(values);
    fclose(fp);

    /* Format Column Labels */
    XtVaSetValues(table,
        XmNxrtTblContext,    XrtTblSetContext(XRTTBL_LABEL, XRTTBL_ALL),
        XmNxrtTblBackgroundContext,  "blue",
        XmNxrtTblForegroundContext,  "white",
        XmNxrtTblAlignmentContext,   XRTTBL_ALIGNMENT_CENTER,
        NULL);
```

```
    /* Change color of President's Salary */
    XtVaSetValues(table,
        XmNxrtTblContext,    XrtTblSetContext(0, 4),
        XmNxrtTblForegroundContext,  "red",
        NULL);

    /* Add a title */
    title = XtVaCreateWidget("title",
        xtXrtLabelWidgetClass,       table,
        XmNxrtLblBorderType,         XRTLBL_BORDER_SHADOW,
        XmNlabelString,
            (s = XmStringCreateSimple("Employee Listing")),
        XmNxrtTblLocation,           XRTTBL_LOCATION_TOP,
        XmNxrtTblTopOffset,          10,
        XmNxrtTblBottomOffset,       20,
        XtVaTypedArg,                XmNfontList,
        XmRString, "-*-times-bold-r-*-*-*-180-*-*-*-*-iso8859-1",
        strlen("-*-times-bold-r-*-*-*-180-*-*-*-*-iso8859-1") +1,
        NULL);

    XmStringFree(s);
    XtRealizeWidget(toplevel);

    XtMainLoop();
}
```

Index

2-byte fonts 137

A

action routines 225
  CancelEdit() 225
  CommitEdit() 225
  Drag() 225
  NoAction() 225
  ResizeCell() 92, 226
  Scroll() 89, 226
  Select() 103, 226
  TraverseNext() 227
  TraversePrev() 227
  TraverseToCell() 94, 227
alignment, cell values 55
ANSI C, compatibility with XRT/table 13
arglist
  using 113
Assumptions 5
authorizing
  shared libraries 13
authorizing a program 12

B

basic concepts
  cell 23
  Context resources 26
  current cell 23
  label 23
  Series resources 26
blinking
  duration of a flash 64
  setting for a context 64
  stopping for a context 64
bombhunt demo 280
border sides, specifying 57
border type, specifying 56
border width, specifying 56
border, frame 57

C

C++
  compatibility with XRT/table 13
  header file 142
  use of 142
C++ classes
  XrtTable 142
callback resources, summary 39
CancelEdit() action 225
Cell Sensitivity 104
cell values
  clearing table 81
  getting into XrtTblCallValues 76
  handling dynamic data 76
  loading from a file 15
  maintaining multiple structures 128
  retrieving 78
  setting 77
cell, definition of 23
cell/label Context resources, summary 35
cell/label Series resources, summary 37
CELL_VALUE macro 74, 78, 221
CELL_VALUE_EXISTS macro 221
cell-only resources, summary 36, 37
CellValue callback 82, 151
changes in this release 3
Changing Default Pointers 105
character method of row height/column width 48
child widget resources, summary 38
class hierarchy 147
class information 147
class name 147
class pointer 147
  XrtLabel 271
clip pixmap 60
cloned widgets 110
color resources, setting 29
colors
  background 53
  child widgets 53
  colormap flashing 30
  dynamic 30
  foreground 53
  pixel values 53
  repeating 54
  selected cells 53
  shadow 54
  static 30
column width
  specifying 48
columns
  flexible 51
  setting number frozen 41

setting number of 40
setting number visible 41
combo boxes
using 97
CommitEdit() action 225
compiling and authorizing a program 12
Compound String resources 29
Compound Strings
creating and using in a table 124
constraint resources, summary 38
Context resources
definition of 26
getting 27
resetting 27
setting 26
ContextValue callback 139, 154
CreateWidget callback 113, 155
adding 114
current cell, definition of 23
cursor positioning 96
customizing XRT/table warnings and errors 142

D

data type listing 229
data validation
code examples 118
extending data types 117
use of ValidateCell callback 116
debugging XRT/table programs 30
default translations 223
definitions
cell 23
Context resources 26
current cell 23
label 23
Series resources 26
XmText widget as current cell 24
demo programs, discussion of 280
Destroying a Widget 113
Disabling Row/Column Labels Dragging 108
displaying entire table 41
DisplayWidget callback 114, 159
double-clicks, handling 88
drag and drop 106, 130
Drag() action 225
dynamic colors 30

E

editing cells
disallowing 96
XmText widget resources 98
EnterCell callback 98, 160
entry.c example 279
environment variables, XRTHOME 12
error messages 30
customizing 142
event table 223
example programs, listing of 279
executable, compiling and authorizing 12
Export callback 161
Expose callback 123, 163
extending data types 117

F

fallback resources
using Series resources 127
Fields in Resource Files 120
flashing
duration of flash 64
setting for a context 64
stopping for a context 64
flexible rows and columns 51
fonts
2-byte 137
specifying 55
use of ISO Latin1 137
form demo 280
frame
border 57
shadow thickness 58
freezing rows/columns 41
frozen columns
use of XrtTblMoveColumns() 42
frozen rows
use of XrtTblMoveRows() 42
frozen rows/columns
placement on screen 42

G

getting started with XRT/table 11
graph_editor demo 280

H

header file 147
hiding rows/columns 51

I

Import callback 166
improving performance 137
managing context values 139
include file 147
include file, XrtLabel 271
Index 283
inherited resources 190
XrtLabel 277
instantiating table widget 147
internationalization of tables 137
introduction
loading cell values 15

specifying label text 16
typographical conventions 5
introduction to XRT/table widget 11
IS_CELL macro 221
IS_COLUMNLABEL macro 221
IS_LABEL macro 221
IS_ROWLABEL macro 221

L

label, definition of 23
LABEL_VALUE macro 44, 222
LABEL_VALUE_EXISTS macro 222
LabelCreate callback 45, 170
label-only resources, summary 36
labels
creation callbacks 45
placement on screen 44
retrieving 43
spacing from cell area 44
specifying 16, 43
specifying text type 78
learning XRT/table 11, 20
linking
shared libraries 13
list display mode 88
list.c example 279

M

macros 221
CELL_VALUE 221
CELL_VALUE_EXISTS 221
IS_CELL 221
IS_COLUMNLABEL 221
IS_LABEL 221
IS_ROWLABEL 221
LABEL_VALUE 222
LABEL_VALUE_EXISTS 222
managing context values 139
limitations 141
miscellaneous resources, summary 39
monochrome X Servers, use of 30
Motif 2.0 restrictions 79
mouse pointers
default 104
disabling tracking of 105
multiply.c example 279
mvc demo 280

N

new features in this release 2
NoAction() action 225
disabling translations 87
number of rows/columns, setting 40

O

overflow
compared with spanning 61
specifying 60

P

performance improvement 137
managing context values 139
Series resources 127
pixel method of row height/column width 48
pixmap
clip 60
formats supported 58
layout in cell 59
removing 59
specifying 58
suppressing shadow 59
play demo 280
pointer resources 28
pointers
changing default 105
PostScript
output procedures 65
specifying substitution font 68
Preview Scrolling 90
border 58
preview scrolling
borders 56
Print callback 69, 175
printing 65
limitations 66
XrtLabel 275
PrintWidget callback 71, 176
Product Feedback and Announcements 6
programming Series resources 126

R

repeating colors 54
Resize callback 123, 177
ResizeCell callback 93, 177
ResizeCell() action 92, 226
resizing
approaches to 122
resizing rows/columns
disabling 92
ResizeCell callback 93
resource files
example 262
syntax 257
use of 257
resource type/class listing 249
resources
affecting table resizing 123
callback 39
cell/label, Context 35

cell/label, Series 37
cell-only, Context 36
cell-only, series 37
child widget 38
constraint 38
inherited 190
label-only 36
other 39
reading from files 143
scrollbar 38
summary of 33
writing to files 143
row height
specifying 48
rows
flexible 51
setting number frozen 41
setting number of 40
setting number visible 41

S

sample code 279
Scroll callback 92, 179
Scroll() action 89, 226
scrollbar resources, summary 38
scrollbars
standard behavior 89
XmScrolledList behavior 89
scrolling
by an application 91
disabling 91
double buffering 138
Select callback 102, 179
Select() action 103, 226
selecting cells 95, 99
selection
removing a range 102
removing all ranges 102
Series
components of 126
convenience procedures 126
Series resources
as fallback resources 127
definition of 26
programming 126
setting for several contexts 127
setting in resource files 28
to improve performance 127
shadow thickness 58
shared libraries
linking and authorizing 13
simple.c example 279
discussion of 13
listing of 279, 281
simple.dat file 16
simple_cpp.cxx example 279
Sort callback 182
spanning
adding a range 63
compared with overflow 61
removing a range 63
removing all ranges 63
specifying colors 29
spin boxes
using 97
spreadsheet demo 280
static colors 30
stocks demo 280
styles example 279
summary of resources 33
surface_editor demo 280
SYLK 83, 220

T

table data
clearing entire table 81
getting into XrtTblCellValues 76
handling dynamic data 76
retrieving 78
setting cell values 77
table frame 58
border 57
table printing 65
limitations 66
print callbacks 69
print widget callbacks 71
table resizing
approaches to 122
XmNallowShellResize 123
table size, determining 122
table_entry demo 280
technical support 5
title
adding to table 46
determining coordinates of 48
location and offset 46
removing from table 48
rotating text of 48
use of XrtLabel widget 46
TrackCursor callback 105, 183
translations
disabling 87
setting 87
translations/actions, listing of 223
TraverseCell callback 95, 184
TraverseNext() action 227
TraversePrev() action 227
TraverseToCell() action 94, 227
traversing
by an application 95
disallowing by users 94
typographical conventions 5

U

UIL
  arguments 265
  example code 143
  header file 143
  reasons 265
  use of 142
user interaction
  customizing 87
  default 85
  double-clicks 88
Using an ArgList 113
Using Combo Boxes and Spin Boxes 97

V

ValidateCell callback 118, 185
  data validation 116
validation of cell entry 116
validation policy 117
visible rows/columns
  displaying all in table 41
  setting number of 41

W

warning messages 30
  customizing 142
widget
  basic concepts 21
  class hierarchy 147
  class name 147
  class pointer 147
  destroying 113
  include file 147
  instantiating 147
  introduction 1
  managed by XRT/table 24
  Motif class hierarchy 22
  Motif integration 21
  synopsis 147
widget cloning 110
  widgets that can be cloned 111
widget in cell/label
  adding 110
  code examples 115
  location 113
  removing 113
  setting resources on 111
  using variable rows/columns 112
widgets demo 280
WidgetSetValue callback 114, 187
word wrapping 61

X

XCreateFontCursor() 105
XmCreateXrtTable() 191
XmNallowShellResize, relationship to table resizing 123
XmNbottomShadowColor 58
XmNdragInitiatorProtocolStyle 107
XmNdragReceiverProtocolStyle 107
XmNfontList 176
XmNhighlightThickness
  relationship with margin height/width 171
XmNhighlightThickness, relationship to margins 52
XmNlabelPixmap 273
XmNlabelPixmap XrtLabel resource 276
XmNlabelString 273
XmNlabelString XrtLabel resource 276
XmNlabelType 272
XmNmodifyVerifyCallback
  use of by XRT/table 150
XmNmultiClickTime, use of with Select callback 88
XmNshadowThickness
  use with XrtLabel 273
XmNshadowThickness XrtLabel resource 276
XmNstringDirection
  relationship with text alignment 148
XmNtopShadowColor 58
XmNwordWrap 188
XmNxrtFldMask 121
XmNxrtFldMatchPickList 121
XmNxrtFldPickList 121
XmNxrtFldPickListIndex 121
XmNxrtLblBackgroundString 274, 276
XmNxrtLblBorderType 273, 276
XmNxrtLblForegroundString 274, 276
XmNxrtLblRotation 48, 274, 277
XmNxrtTblAlignmentContext 55, 148, 229
XmNxrtTblAlignmentSeries 148
XmNxrtTblAllowCellResize 92, 148
  effect on mouse pointers 104
XmNxrtTblAllowResize 122, 148
XmNxrtTblArgListContext 113, 148
XmNxrtTblArgListSeries 148
XmNxrtTblBackgroundContext 53, 54, 149
XmNxrtTblBackgroundPixelContext 53, 149
XmNxrtTblBackgroundSeries 149, 177
XmNxrtTblBorderSidesContext 57, 149
XmNxrtTblBorderSidesSeries 149
XmNxrtTblBorderTypeContext 56, 57, 150
XmNxrtTblBorderTypeSeries 150
XmNxrtTblBottomOffset 47, 189
XmNxrtTblBottomShadowContext 54, 150
XmNxrtTblCaseContext 96, 150
XmNxrtTblCaseSeries 150
XmNxrtTblCellValueCallback 29, 61, 82, 141, 151, 172
XmNxrtTblCellValueContext 29, 77, 78, 141, 151
XmNxrtTblCellValueFunc 151, 172
  using to improve performance 139
XmNxrtTblCellValues 152

XmNxrtTblCharHeightContext 49, 152, 173
XmNxrtTblCharHeightSeries 152
XmNxrtTblCharWidthContext 49, 152, 173
XmNxrtTblCharWidthSeries 152
XmNxrtTblChildrenPersistent 152
XmNxrtTblClipChildren 87, 152
XmNxrtTblClipChildrenPersistent 152
XmNxrtTblClipPixmapContext 60, 153
XmNxrtTblClipPixmapSeries 153
XmNxrtTblColumnLabelOffset 44, 153
XmNxrtTblColumnLabelPlacement 44, 153
XmNxrtTblColumnLabels 29, 43, 153
XmNxrtTblContext 153, 211
XmNxrtTblContextValueCallback 138, 139, 154
XmNxrtTblCreateWidgetCallback 113, 155
XmNxrtTblCurrencyPickListContext 156
XmNxrtTblCurrencyPickListSeries 156
XmNxrtTblCursorCorner 156
XmNxrtTblCursorHorizontal 156
XmNxrtTblCursorNontraversable 156
XmNxrtTblCursorTraversable 156
XmNxrtTblCursorVertical 156
XmNxrtTblDataTypeContext 116, 119, 156, 164, 197
XmNxrtTblDataTypeList 117, 157
XmNxrtTblDataTypeSeries 158
XmNxrtTblDatePickListContext 158
XmNxrtTblDatePickListSeries 159
XmNxrtTblDebug 30, 159
XmNxrtTblDisplayClipArrows 60, 159
XmNxrtTblDisplayHorizScrollBar 90, 91, 159
XmNxrtTblDisplayVertScrollBar 90, 91, 159
XmNxrtTblDisplayWidgetCallback 114, 159
XmNxrtTblDoubleBuffer 138, 160
XmNxrtTblEditableContext 96, 108, 160
XmNxrtTblEditableSeries 160
XmNxrtTblEnterCellCallback 98, 160, 214, 227, 237
XmNxrtTblExportCallback 133, 161, 237, 238
XmNxrtTblExportDelimiter 133, 162, 168
XmNxrtTblExportTargets 130, 162, 238
XmNxrtTblExposeCallback 123, 163
XmNXrtTblFieldArgList 120
XmNxrtTblFieldArgListContext 120, 121, 164
XmNxrtTblFieldArgListSeries 164
XmNxrtTblFloatPickListContext 164
XmNxrtTblFloatPickListSeries 164
XmNxrtTblFontListContext 55, 141, 164
XmNxrtTblFontListSeries 164
XmNxrtTblForegroundContext 53, 164
XmNxrtTblForegroundPixelContext 53, 165
XmNxrtTblForegroundSeries 165, 177
XmNxrtTblFrameBorderType 57, 165
XmNxrtTblFrameShadowThickness 58, 165
XmNxrtTblFrozenColumnPlacement 42, 165
XmNxrtTblFrozenColumns 41, 166
XmNxrtTblFrozenRowPlacement 42, 166
XmNxrtTblFrozenRows 41, 166
XmNxrtTblHorizontalScrollBar 91, 166
XmNxrtTblHorizScrollBarLeftAttachment 89, 166
XmNxrtTblHorizScrollBarPlacement 90, 166
XmNxrtTblHorizScrollBarPosition 89, 166, 182
XmNxrtTblHorizScrollBarRightAttachment 89, 166
XmNxrtTblImportCallback 134, 166
XmNxrtTblImportDelimiter 132
XmNxrtTblImportTargets 108, 130, 168, 238
XmNxrtTblIntPickListContext 169
XmNxrtTblIntPickListSeries 169
XmNxrtTblJumpScroll 90, 169
XmNxrtTblLabelContext 29, 43, 169
XmNxrtTblLabelCreateCallback 29, 45, 61, 170, 172
XmNxrtTblLabelCreateFunc 170, 172
XmNxrtTblLeftColumn 89, 91, 170
XmNxrtTblLeftOffset 47, 189
XmNxrtTblLocation 46, 48, 189
XmNxrtTblMarginHeight 52, 170
XmNxrtTblMarginWidth 52, 171
XmNxrtTblMaxLengthContext 97, 171
XmNxrtTblMaxLengthSeries 171
XmNxrtTblMinCellVisibility 95, 171
XmNxrtTblMinFlexibleHeight 51, 171, 173
XmNxrtTblMinFlexibleWidth 51, 171, 173
XmNxrtTblMode 88, 171
XmNxrtTblMultilineContext 97, 172
XmNxrtTblMultilineSeries 172
XmNxrtTblNumColumns 40, 41, 79, 80, 172
XmNxrtTblNumRows 40, 41, 79, 80, 172
XmNxrtTblOverflowContext 60, 141, 172
XmNxrtTblOverflowSeries 172
XmNxrtTblPixelHeight
  user row resizing 92
XmNxrtTblPixelHeightContext 50, 51, 152, 171, 172
XmNxrtTblPixelHeightSeries 173
XmNxrtTblPixelWidth
  user column resizing 92
XmNxrtTblPixelWidthContext 50, 51, 152, 171, 173
XmNxrtTblPixelWidthSeries 173
XmNxrtTblPixmapContext 58, 173
XmNxrtTblPixmapLayoutContext 59, 174, 239
XmNxrtTblPixmapLayoutSeries 174
XmNxrtTblPixmapSeries 174
XmNxrtTblPixmapShadowContext 59, 174
XmNxrtTblPixmapShadowSeries 174
XmNxrtTblPreviewScrolling 90, 174
XmNxrtTblPrintCallback 69, 175, 240
XmNxrtTblPrintWidgetCallback 71, 176, 241
XmNxrtTblPSFontMapList 68, 176
XmNxrtTblRangeContext 154, 176
XmNxrtTblRepaint 177
XmNxrtTblRepeatBackgroundColors 54, 149, 177
XmNxrtTblRepeatForegroundColors 54, 177
XmNxrtTblResizeCallback 123, 177, 243
XmNxrtTblResizeCellCallback 93, 177, 243
XmNxrtTblResizeToCell 189
XmNxrtTblRightOffset 47, 189
XmNxrtTblRotationContext 178
XmNxrtTblRowLabelOffset 44, 178

XmNxrtTblRowLabelPlacement 44, 178
XmNxrtTblRowLabels 29, 43, 178
XmNxrtTblScrollCallback 92, 179, 244
XmNxrtTblSelectCallback 53, 102, 179, 181, 214, 226, 244
XmNxrtTblSelectedBackground 53, 180
XmNxrtTblSelectedCellList 100, 180
XmNxrtTblSelectedForeground 53, 180
XmNxrtTblSelectionPolicy 99, 106, 181, 226
XmNxrtTblSensitiveContext 104, 181
XmNxrtTblSensitiveSeries 181
XmNxrtTblSetTextCursorPos 96, 181
XmNxrtTblShadowThickness 54, 56, 138, 181
XmNxrtTblSortCallback 182, 213
XmNxrtTblSpace 90, 182
XmNxrtTblSpanList 62, 182
XmNxrtTblStringPickListContext 121, 182
XmNxrtTblStringPickListSeries 182
XmNxrtTblText 98
XmNxrtTblTextInitSelect 95, 183
XmNxrtTblTopOffset 47, 189
XmNxrtTblTopRow 89, 91, 183
XmNxrtTblTopShadowContext 54, 183
XmNxrtTblTrackCursor 105, 183
  using to improve performance 138
XmNxrtTblTrackCursorCallback 105, 183
XmNxrtTblTraversableContext 94, 138, 141, 160, 184
  effect on mouse pointers 104
XmNxrtTblTraversableSeries 184
XmNxrtTblTraverseCellCallback 95, 138, 184, 227, 245
XmNxrtTblUserDataContext 65, 184, 185
XmNxrtTblUserDataSeries 184
XmNxrtTblValidateCellCallback 118, 185, 194, 246
XmNxrtTblValidatePolicy 117, 185
XmNxrtTblVerticalScrollBar 91, 186
XmNxrtTblVertScrollBarBottomAttachment 89, 186
XmNxrtTblVertScrollBarPlacement 90, 186
XmNxrtTblVertScrollBarPosition 89, 182, 186
XmNxrtTblVertScrollBarTopAttachment 89, 186
XmNxrtTblVisibleColumns 41, 186
XmNxrtTblVisibleRows 41, 186
XmNxrtTblWidgetClassList 115, 187
XmNxrtTblWidgetContext 110, 113, 187
XmNxrtTblWidgetLocation 110, 189
XmNxrtTblWidgetSeries 187
XmNxrtTblWidgetSetValueCallback 114, 187, 247
XmNxrtTblWidgetShadowContext 115, 187
XmNxrtTblWidgetShadowSeries 187
XmNxrtTblWordWrapContext 61, 188
XmNxrtTblWordWrapSeries 188
XmNxrtTblXmStringGetMode 44, 78, 188
XmNxrtTblXmStringSetMode 44, 77, 188
XmStringCreateLtoR() 195
XmStrings 124
XmText, use of as current cell 24
XmTextSetSelection() 183
XPM pixmap format 58
XrmMergeDatabases() 142
XRT/3d
  using datasets in table 130
XRT/field, using with XRT/table 119
XRT/graph
  using datasets in table 130
XRT/label, see XrtLabel
XRT/table
  basic concepts 21
  changes in this release 3
  class hierarchy 147
  class information 147
  class name 147
  class pointer 147
  include file 147
  inherited resources 190
  instantiating 147
  integration with Motif 21
  introduction 1
  list display mode 88
  Motif class hierarchy 22
  new features 2
  resource types/classes 249
  using resource files 257
  using XRT/field 119
  widgets managed 24
xrt_auth program
  flags 12
  use of 12
XRTHOME environment variable 12
XrtLabel
  class hierarchy 271
  class name 271
  class pointer 271
  creating 272
  include file 271
  inherited resources 277
  instantiating 271
  label type 272
  printing 275
  use as XRT/table title 275
  visual components 272
  widget synopsis 271
XrtTblAction 229
XrtTblAddCallback() 191
XrtTblAddColumns() 75, 172, 192
XrtTblAddRows() 75, 79, 172, 192
XrtTblAlignment 229
XrtTblAnyValue structure 126, 229
XrtTblArgListValue structure 230
XrtTblAttachment 230
XrtTblBooleanValue structure 230
XrtTblBorderSides 230
XrtTblBorderType 230
XrtTblCancelEdit() 193
XrtTblCase 231
XrtTblCellValueCallbackStruct structure 151, 231
XrtTblCellValues structure 231
  convenience macros 74
  convenience procedures 74
  description of 73

using more than one 128
XrtTblCellValuesCopy() 75, 193
XrtTblCheckDispatcher() 193
XrtTblCheckIsBoolean() 117, 193
XrtTblCheckIsDouble() 117, 194
XrtTblCheckIsEmpty() 194
XrtTblCheckIsFloat() 117, 194
XrtTblCheckIsInt() 117, 194
XrtTblColorValue structure 231
XrtTblColumnPlacement 231
XrtTblCommitEdit() 194, 214
XrtTblCompareStruct structure 232
XrtTblContext structure 232
XrtTblContextReason 232
XrtTblContextValue structure 233
XrtTblContextValueCallbackStruct structure 154, 233
XrtTblCreateSpreadsheetLabel() 46, 195
XrtTblCreateWidgetCallbackStruct structure 155, 234
use of parameters 113
XrtTblCvtPixelToString() 195
XrtTblCvtStringToXmString() 195
XrtTblCvtXmStringToString() 196
XrtTblDataType 234
XrtTblDataTypeRoutine structure 234
XrtTblDeleteColumns() 75, 80, 172, 196
XrtTblDeleteRows() 75, 80, 172, 196
XrtTblDestroyCellValues() 196
XrtTblDestroyImportExportList() 203
XrtTblDisplayWidgetCallbackStruct structure 160, 235
use of parameters 114
XrtTblDispScrollBar 235
XrtTblDoubleBuffer 235
XrtTblDrawPS() 65, 197
internationalization support 137
use with XrtLabel 275
XrtTblDrawPSarg 236
XrtTblEnableXrtField() 119, 197
XrtTblEnterCellCallbackStruct structure 161, 237
description of parameters 98
XrtTblEventToRowCol() 88, 197
XrtTblExportCallbackStruct structure 161, 237, 238
XrtTblExposeCallbackStruct structure 163, 237
XrtTblFlash() 63, 64, 198
XrtTblFontListValue structure 237
XrtTblGet3dData() 130, 198
XrtTblGetCell() 75, 78, 198
XrtTblGetCellDimensions() 50, 51, 199
XrtTblGetCellValues() 75, 171, 199
XrtTblGetCellValuesSize() 199
XrtTblGetCurrentCell() 200
XrtTblGetGraphData() 130, 200
XrtTblGetPreviewProperties() 91, 200
XrtTblGetRowColByWidget() 113, 200
XrtTblGetRowColLocation() 201
XrtTblGetSelectedRange() 101, 180, 201
XrtTblGetSpanByRowCol() 63, 201
XrtTblGetWidgetByRowCol() 202
XrtTblGetXrtFieldByType() 121, 202
XrtTblImportCallbackStruct structure 167
XrtTblImportExportListAdd() 202
XrtTblImportExportListCopy() 202
XrtTblImportExportListDelete() 202
XrtTblImportExportStruct structure 238
XrtTblIntValue structure 238
XrtTblIsCellSelected() 101, 203
XrtTblIsCellTraversable() 203
XrtTblIsCellVisible() 91, 203
XrtTblJumpScroll 238
XrtTblLabelCreateCallbackStruct structure 170, 238
XrtTblLocation 239
XrtTblMakeCellVisible() 92, 204
XrtTblMode 239
XrtTblMoveColumns() 75, 80, 204
XrtTblMoveRows() 75, 80, 204
XrtTblMrmInitialize() 143, 204
XrtTblOverflow 239
XrtTblPicklistValue 239
XrtTblPixmapLayout 239
XrtTblPixmapValue structure 240
XrtTblPointerValue structure 240
XrtTblPreviewScroll 240
XrtTblPrintCallbackStruct structure 70, 175, 240
XrtTblPrintWidgetCallbackStruct structure 176, 241
XrtTblPSarg structure 241
XrtTblPSFontMap structure 241
XrtTblRange structure 241
description of parameters 100
XrtTblRangeContext 113
XrtTblRangeListAdd() 63, 101, 205
XrtTblRangeListCopy() 205
XrtTblRangeListDelete() 63, 102, 205
XrtTblRangeListFind() 102, 205
XrtTblRangeListFindRange() 102, 206
XrtTblRangeListFree() 206
XrtTblReadAscii() 206
use of in resource files 259
XrtTblReadResources() 144, 206
XrtTblReason 242
XrtTblRemoveAllCallbacks() 207
XrtTblRemoveCallback() 207
XrtTblRepaint 242
XrtTblRepaintRange() 207
XrtTblResize 243
XrtTblResizeCallbackStruct structure 177, 243
XrtTblResizeCellCallbackStruct structure 178, 243
use of parameters 93
XrtTblRestoreAllCursors() 105, 208
XrtTblRowPlacement 243
XrtTblSBPosition 243
XrtTblScrollCallbackStruct structure 179, 244
XrtTblSelectCallbackStruct structure 179, 244
use of parameters 102
XrtTblSelectionPolicy 244
XrtTblSeries structure 244
description of 126
XrtTblSeriesAdd() 126, 208
XrtTblSeriesAddValue() 126, 208
XrtTblSeriesCopy() 126, 208
XrtTblSeriesCreate() 126, 208

XrtTblSeriesDelete() 126, 209
XrtTblSeriesDestroy() 127, 209
XrtTblSeriesFind() 126, 128, 209
XrtTblSeriesIterate() 126, 127, 209
XrtTblSeriesSetValue() 126, 210
XrtTblSetAllCursors() 105, 210
XrtTblSetCell() 75, 77, 139, 210
XrtTblSetCellValues() 75, 152, 171, 211
XrtTblSetContext() 27, 154, 211
XrtTblSetPreviewProperties() 91, 211
XrtTblSetPSarg() 212
XrtTblSetRange() 27, 176, 212
XrtTblSetReverseConverters() 212
XrtTblSetXmStringConverter() 195, 212
XrtTblSortCallbackStruct structure 182, 245
XrtTblSortTable() 75, 213, 232
XrtTblStringValue structure 245
XrtTblSyncScrolling() 213
XrtTblTrackCursorCallbackStruct structure 183, 245
  description of parameters 105
XrtTblTraverseCellCallbackStruct
  use of parameters 95
XrtTblTraverseCellCallbackStruct structure 184, 245
XrtTblTraverseToCell() 95, 214
XrtTblVaDrawPS() 65, 214
  internationalization support 137
  use with XrtLabel 275
XrtTblValidateCellCallbackStruct structure 118, 185, 246
XrtTblValidatePolicy 246
XrtTblValueType 246
XrtTblValueType structure 126
XrtTblWidgetClass structure 247
XrtTblWidgetPos structure 247
XrtTblWidgetSetValueCallbackStruct structure 187, 247
XrtTblWidgetValue structure 247
XrtTblWriteAscii() 83, 219
XrtTblWriteResources() 143, 219
XrtTblWriteSylk() 83, 220
XrtTblXmStringGet 247
XrtTblXmStringList structure 17, 248
XrtTblXmStringProc 248
XrtTblXmStringSet 248
XrtTblXYtoRowCol() 220
Xt Error database
  customizing 142
  location 142
  modifying messages 142
XtAppWarningMsg() 142
XtIsXrtTable() 220
XtQueryGeometry() 112
XtXrt3d 22
XtXrtGraph 22
XtXrtLabel 271, 272
xtXrtLabelWidgetClass 271
XtXrtTable 22, 147
xtXrtTableWidgetClass 147